Stony Brook University

Task Scheduling in Modern Data Center: Task Placement and Resource Allocation

A Dissertation Presented

by **Li Shi**

to

The Graduate School

in Partial Fulfillment of the requirements

for the Degree of

Doctor of Philosophy

in

Computer Engineering

Stony Brook University

December 2016

Stony Brook University

The Graduate School

Li Shi

We, the dissertation committee for the above candidate for the

Doctor of Philosophy degree, hereby recommend

acceptance of this dissertation

Thomas G. Robertazzi
Professor in Department of Electrical and Computer Engineering

Wendy Tang
Associate Professor in Department of Electrical and Computer Engineering

Yue Zhao
Assistant Professor in Department of Electrical and Computer Engineering

Esther Arkin
Professor in Department of Applied Mathematics and Statistics

This dissertation is accepted by the Graduate School

Nancy Goroff
Interim Dean of the Graduate School

Abstract of the Dissertation

Task Scheduling in Modern Data Center: Task Placement and Resource Allocation

by

Li Shi

Doctor of Philosophy

in

Computer Engineering

Stony Brook University

2016

In modern data centers, the wide use of virtualization techniques has enabled dynamic resource allocation in the form of virtual machines and virtual networks. With such an ability, scheduling tasks, including both computing tasks and data transfer tasks, comprises (a) placing tasks on servers or network paths and (b) allocating a certain amount of resources to each task. In such a context, minimizing the completion time of the tasks, as a critical goal on many task processing platforms, requires joint consideration of both task placement and resource allocation. While many approaches have been proposed in the area of scheduling tasks in data centers, few of them consider

the two factors together, which lead the inefficiency of these approaches. In this dissertation, we study the problem of task scheduling in data centers and propose solutions that jointly consider task placement and resource allocation. We start from a fundamental problem: how to optimally allocate resource according to determined task placements. We formulate this problem as a convex optimization problem and develop an analytical solution. Based on the solution of this problem, we further study three more complex problems: (a) Energy-aware scheduling of embarrassingly parallel jobs and resource allocation in cloud; (b) Coflow scheduling in data centers: routing and bandwidth allocation; (c) Scheduling of independent flows in data centers: routing and bandwidth allocation. Each of these problems is formulated as a Non-linear Mixed Integer Programming problem. Offline algorithms and online schedulers that jointly consider task placement and resource allocation are proposed to solve these problems. We compare the proposed solutions with existing approaches through simulations and demonstrate the superior performance of the proposed solutions.

Contents

List of Figures

List of Tables

Acknowledgements

I would like to express my sincere gratitude to really a lot of people. Without their guidance, their help, and their support, I would not be able to finish this dissertation. Here I can only mention a small part of them.

First and foremost, I would like to express my deepest appreciation to my advisor Prof. Thomas G. Robertazzi for the continuous support of my Ph.D study and related research, for his patience, motivation, and immense knowledge. He taught me how to do research and gave me the largest and the most important support when I was in the hardest period of my PhD study. Without his help and support, it would not possible for me to pursue my PhD degree. He gave me a lot of freedom on selecting my research direction, respected to any of my research thoughts, but always provided very many valuable comments and suggestions. His guidance helped me in all the time of research and writing of this dissertation. He is definitely the most easygoing and the best advisor. I sincerely appreciate all his efforts putting on me and am really lucky and honored to be his student!

Besides my advisor, I would like to thank the rest of my defense committee: Prof. Wendy Tang, Prof. Yue Zhao, and Prof. Esther Arkin, for their insightful comments and encouragement and also for the questions which inspired me to widen my research from various perspectives.

I would like to also give my sincerest gratitude to my love and my wife Pengmei Zhu. She gave me tremendous support during my whole PhD study. She understood the hardness of doing research work and never complained that I put so much time on my research, which I always feel sorry about. She

also provided her special insight on many problems I was facing, which were always constructive and helpful. She gave me so much power, motivation and encouragement without which I would never be able to finish my PhD study. She is the kindest, smartest, the most lovely and tolerant women I have ever met. I am so lucky, grateful, and honored to have her as my wife and I love her forever!

I also want to give my deepest appreciation to my parents: Yansheng Shi and Minfang Peng. They gave me huge support both spiritually and financially and cherished with me every achievement I have ever made. I know that they were always worried about my life and study abroad and missed me so much, but they never gave me any pressure but encouraged my all the time. I love them as much as they love me. Sincerely wish them happy and health forever!

Lastly, I thank my friends, Yang Liu, Zheming Zhang, Zhenzhou Peng, Tan Li, Shun Yao, Yufei Ren, Shuchu Han, Hao Wang, Linfeng Gao, Qing Sun, Zupan Hu. Their invaluable suggestions and encouragement helped me pursue my PhD degree and work on this dissertation. I am very grateful to each of them. In addition, I thank Professor Petar M. Djuric for his encouragement and for providing the space and facilities. This important support has made the laboratory very convenient and suitable for research work.

Chapter 1

Introduction

In recent decades, high performance computers, high speed networks and large-scale storage systems have changed the world. A huge amount of data can be collected via high speed networks, stored in large-scale storage systems and processed by high performance computers. By performing millions times of computing on such a large volume of data every second, we can test a hypothesis, solve a complex mathematical problem, or find a trend hidden behind the input data in a short time. This kind of data processing has been applied on many different areas, like physics, biology, medicine, manufacturing, advertising, and finance, and has greatly changed the world we live.

In such a context, improving the processing capability is always a critical goal. While scientists and engineers keep developing better individual devices, for example, processors with higher clock frequency and network cable with higher bandwidth, uniting multiple machines together and making them work in parallel is another important way invented to improve the

processing capability. Usually, we call such a processing system as a parallel/distributed computing system which is composed of computing nodes (processors/servers) that are interconnected by networks. User tasks are executed simultaneously on these nodes and input/output data are moved among these nodes through the network. Examples of such parallel computing system include computing clusters, computing grids and recently data centers.

In a parallel computing system, it is common that multiple user tasks are waiting to be executed and a scheduling system is required to schedule those tasks. Those tasks can be computing tasks executed on computing nodes and can also be data transfer tasks which should be routed through some paths. Some of those tasks can belong to the same user or the same user job, in which case the scheduling objective can be minimizing the makespan of those tasks. On the other hand, some other tasks can belong to different users or different user jobs, in which case the scheduling objective can be minimizing the total execution time of those tasks. How to efficiently schedule those tasks in different types with different objectives is a critical problem to solve in the parallel computing environment.

In traditional platforms, like computing clusters and grids, we usually consider the resources used to execute user tasks as static or consider the way of allocating resources to user tasks as static. For example, computing tasks place on the same computing node are usually sequentially executed [1] and hence each task gets the whole available computing resources on that node when the task is executed. For another example, when executing data transfer tasks, i.e., transferring data from the source to the destination of

those tasks along some selected paths, the available bandwidth of physical links in the network are shared between data transfers (i.e., flows) in a best efforts manner. In such traditional parallel computing systems, task scheduling is more about determining a task placement with a specific objective. Such task scheduling problem has been well studied [1–12].

However, in modern data centers, the task scheduling problem becomes much more complex because of the wide use of modern virtualization techniques which has enabled dynamic resource allocation. Through virtualization techniques, a data center administrator is able to dynamically allocate computing resource to tasks in the form of Virtual Machines (VMs) with a flexible amount of computing resources while keeping the provisioned VMs isolated and interference free from each other. In this way, computing tasks placed on the same computing node can be simultaneously executed on their own VM with guaranteed share of the computing resources of the underlying computing node. Through virtualization techniques, a data center administrator is also able to dynamically provide specific bandwidth guarantees (i.e., allocated bandwidth) in the form of Virtual Networks to concurrent data transfers that are sharing same links [13–17]. In this way, we can actively prevent bandwidth competition and provide guaranteed performance to these data transfers.

While the advanced technique of dynamic resource allocation brings significant benefits, it also changes the scope of the task scheduling problem which includes both task placement and resource allocation now. In such a new paradigm, how to efficiently schedule tasks is an important but challenging problem. To solve the task scheduling problem, we need to answer

three questions:

1. How to optimally allocate resources to tasks with pre-determined task placement plan?

2. How to place tasks and allocate resources to a set of tasks in an offline scenario?

3. How to address multiple tasks in online scheduling?

Naturally, these three questions are in a progressive relationship. By answering the previous question, we essentially reduce the complexity of the later question. However, when scheduling a large amount of tasks in a large data center in the real world, there exists a vast amount of possible task placement and for each task task placement there are very many ways to allocate resources. Searching the optimal solution in such a huge solution space is not an easy problem to solve. The problem becomes even more complex, when we consider more constraints, like energy consumption, resource usage limitation, heterogeneity of the computing nodes, and etc.

While several approaches have been proposed to schedule either computing tasks or data transfer tasks in data centers [18–30], none of them consider task placement and resource allocation together. Motivated by this, in this paper, we focus on the task scheduling problem in the new environment of modern data centers with joint consideration of task placement and resource allocation. We start from studying a general resource allocation problem in which the task placement has been pre-determined. Subsequently, based on the solution of the general resource allocation problem, we further study three more complex and practical task scheduling problems.

1.1 Our Main Contributions and Dissertation Organization

In this section, we introduce our main contributions made in this dissertation.

In Chapter 2, we study the general problem of optimal resource allocation with pre-determined task placement. In this problem, the task placement for a set of input tasks has been already determined and we are required to allocate the available resources to those tasks. We consider two variants of this problem in which two different (but both common) objectives are considered. We formulate these two variants as convex optimization problems with generalized linear constraints and present an analytical solution of one of the two variants.

In Chapter 3, we focus on the problem of scheduling embarrassingly parallel jobs composed of a set of independent tasks and consider energy consumption during scheduling. Our goal is to determine a task placement plan and a resource allocation plan for such jobs in a way that minimizes the Job Completion Time (JCT). We formulate the problem of scheduling a single job as a Non-linear Mixed Integer Programming problem and present a relaxation with an equivalent Linear Programming problem. We further propose an algorithm named TaPRA and its simplified version TaPRA-fast that solve the single job scheduling problem. Lastly, to address multiple jobs in online scheduling, we propose an online scheduler named OnTaPRA. By comparing with the start-of-the-art algorithms and schedulers via simulations, we demonstrate that TaPRA and TaPRA-fast reduce the JCT by 40%-430% and the OnTaPRA scheduler reduces the average JCT by 60%-

280%. In addition, TaPRA-fast can be 10 times faster than TaPRA with around 5% performance degradation compared to TaPRA, which makes the use of TaPRA-fast very appropriate in practice.

In Chapter 4, we study how to schedule the coflows composed of a set of flows transferring data between two stages of a job. We focus on this problem and jointly consider routing and bandwidth allocation. We formulate the coflow scheduling problem as a Mixed Integer Non-linear Programming problem and present its relaxed convex optimization problem. We further propose two algorithms, CoRBA and its simplified version: CoRBA-fast, that jointly perform routing and bandwidth allocation for a given coflow while minimizes the CCT. Through both offline and online simulations, we demonstrate that CoRBA reduces the CCT by 40%-500% compared to the state-of-the-art algorithms. Simulation results also show that CoRBA-fast can be tens of times faster than all other algorithms with around 10% performance degradation compared to CoRBA, which makes the use of CoRBA-fast very applicable in practice.

In Chapter 5, we focus on scheduling independent flows with the goal of minimizing their total transfer time (TTT) and also jointly consider routing and bandwidth allocation. We first study the problem of scheduling a single set of flows and formulate this problem as a Non-linear Mixed Integer Programming problem. We further present a relaxation with an equivalent convex optimization problem. We propose two algorithms, FRoBA and its simplified version: FRoBA-fast, that jointly perform routing and bandwidth allocation for a given set of flows. Lastly, to address multiple flows in online scheduling, we propose an online scheduler named OnFRoBA whose goal is

to minimize the average Flow Completion Time (FCT). By comparing with the start-of-the-art algorithms and schedulers via simulations, we demonstrate that FRoBA and FRoBA-fast reduce the TTT by 60%-250% and the OnTaPRA scheduler reduces the average FCT by 10%-40%.

In Chapter 6, we discuss about our future work and in Chapter 7, we present our conclusion.

Chapter 2

Optimal Resource Allocation with Pre-determined Task Placement

In this chapter, we study the problem of optimal resource allocation (OptRA) with predetermined task placement. In this problem, a set of tasks have been placed on a set of processing units and we are required to allocate the available resources of the processing units to those tasks with some specific optimization objective. Note that as the first problem studied in this dissertation, the solution of the OptRA problem has critical usage when solving the task placement and resource allocation problem. It essentially reduces the dimension of the task scheduling problem: For any determined task placement plan, we can calculate the corresponding optimal resource allocation plan. In the following, we formally define the OptRA problem.

2.1 Problem Definition

We now introduce input, output, objective function and constraints considered in the problem.

Input. The input contains a set of processing units and a set of tasks. The set of processing units is defined as $\mathcal{P} = \{P_1, P_2, \ldots, P_M\}$ in which P_j is the jth processing units. The available resources of processing unit P_j is denoted by C_{p_j}.

The set of tasks is defined as $\mathcal{T} = \{T_1, T_2, \ldots, T_N\}$ in which T_i is the ith task. We further denote the load of the task T_i by $Load_i$ and assume that the execution time of task T_i on a processing unit with a certain amount of allocated resources r_{t_i}, denoted by et_{t_i}, is

$$et_{t_i} = \frac{Load_i}{r_{t_i}}. \tag{2.1}$$

Output. The output of the OptRA problem contains a resource allocation plan which indicates the amount of resources allocated to each task. We denote denote the resource allocation plan by $\vec{r} = \{r_{t_1}, \ldots, r_{t_N}\}$ in which r_{t_i} is the amount of resources allocated to T_i.

Resource Availability Constraints. When determining the resource allocation plan, we consider the resources availability constraints which are limitations on the total amount of resource that can be allocated to the input tasks on a processing unit or a subset of processing units. Such constraints are usually enforced by system administrator to maintain proper sharing of resources among multiple sets of tasks belong to different users. For example,

there can be limitations on the total amount of compute resources that can be allocated to a user's tasks on a specific rack, or there can be limitations on the total amount of bandwidth that can be allocated to the data transfers belong to a user job on a specific link.

In this dissertation, we focus on those linear resource availability constraints that can be presented as

$$\sum_{i=1}^{N} A_{ik} r_{t_i} \leq B_k,\ k = 1, \ldots, K, \tag{2.2}$$

where A_{ik} is the coefficient of r_{t_i} in the kth constraint; B_k is the total resources allowed to be allocated in the kth constraint.

Objectives. We consider two typical objectives in task scheduling problems and therefore formulates two variants of the OptRA problem:

1. **OptRA-Makespan.** In this problem, our objective is to minimize the makespan of the input tasks, i.e., the completion time of the last finished task. Let the completion time of task T_i be et_{t_i}, based on Equation (2.1), the objective function is

$$Minimize \quad \max_{i=1,\ldots,N} \left\{ et_{t_i} \mid et_{t_i} = \frac{Load_i}{r_{t_i}} \right\}. \tag{2.3}$$

2. **OptRA-Total.** In this problem, our objective is to minimize the overall execution time of the input tasks. Based on Equation (2.1), the objective function is

$$Minimize \quad \sum_{i=1}^{N} et_{t_i} = \sum_{i=1}^{N} \frac{Load_i}{r_{t_i}}. \tag{2.4}$$

In the following, we solve these two variants of the OptRA problem respectively.

2.2 The OptRA-Makespan Problem

Based on the objective and constraints introduced in Section 2.1, we can present the OptRA-Makespan problem as

OptRA-Makespan

Variables

- r_{t_i}: resources allocated to task t_i.
- et_{t_i}: the execution time of task t_i.

Constants

- N: the number of tasks in the set $\mathcal{T}$.
- K: the number of constraints in the problem.
- $Load_i$: the load of task t_i.
- A_{ik}: coefficient of variable r_{t_i} in the kth constraint.
- B_k: the total resources allowed to be allocated in the kth constraint.

Objective:

$$Minimize \quad \max_{i=1,\ldots,N} \left\{ et_{t_i} \mid et_{t_i} = \frac{Load_i}{r_{t_i}}, \right\}. \tag{2.5}$$

Subject to

$$\sum_{i=1}^{N} A_{ik} r_{t_i} \leq B_k,\ k = 1, \ldots, K, \tag{2.6}$$

$$r_{t_i} \geq 0,\ i = 1, \ldots, N. \tag{2.7}$$

We next show that the OptRA-Makespan problem is a convex optimization problem. Because the functions in (2.6) and (2.7) are all affine on r_{t_i} and thus convex, we only need to show the objective function (2.5) is also convex on r_{t_i}. Note that the function et_{t_i} is convex, because its second derivative is nondecreasing when r_{t_i} is larger than 0. Therefore, according to [31], the objective function, i.e., the pointwise maximum function of et_{t_i}, is also a convex function. As a result, the OptRA-Makespan problem is a convex optimization problem.

2.2.1 Analytical Solution and Fast Optimal Algorithm

While existing convex optimization algorithms [31] can be used to solve the OptRA problem, we develop an analytical solution which is more efficient. Specifically, we define a vector $\vec{r}^* = \{r_1^*, r_2^*, \ldots, r_N^*\}$ with

$$r_i^* = \min_{j \in \mathcal{R}_i} \left\{ \frac{L_i}{\sum_{k=1}^{N} A_{kj} L_k} B_j \right\},\ i = 1, \ldots, N, \tag{2.8}$$

where $\mathcal{R}_i$ is the set of constraints in which the coefficient of variable c_{t_i} is not zero. We now show that this vector $\vec{r}^*$ is an optimal solution of the OptRA problem. First of all, we have the following lemma.

Lemma 1. *Assume that for the vector $\vec{r}^*$ defined in Equation 2.8, task p has the largest finish time, i.e., $et_p^* = L_p/r_p^* = \max_{i=1,\ldots,N}\{et_i^*\}$. Also assume that r_p^* obtains the minimum value when constraint u is considered, i.e.,*

$$r_p^* = \min_{j \in \mathcal{R}_p} \left\{ \frac{L_p}{\sum_{k=1}^{N} A_{kj} L_k} B_j \right\} = \frac{L_p}{\sum_{k=1}^{N} A_{ku} L_k} B_u. \tag{2.9}$$

Then every variable c_i that is subjected to constraint u, i.e., $A_{iu} \neq 0$, obtains the minimum value when constraint u is considered, i.e.,

$$r_i^* = \min_{j \in \mathcal{R}_i} \left\{ \frac{L_i}{\sum_{k=1}^{N} A_{kj} L_k} B_j \right\} = \frac{L_i}{\sum_{k=1}^{N} A_{ku} L_k} B_u. \tag{2.10}$$

Proof. To begin with, we assume that there exists a variable r_q^* with $A_{qu} \neq 0$, which gets the minimum value when constraint v (other than constraint u) is considered, i.e.,

$$r_q^* = \min_{j \in \mathcal{R}_q} \left\{ \frac{L_q}{\sum_{k=1}^{N} A_{kv} L_k} B_j \right\} = \frac{L_q}{\sum_{k=1}^{N} A_{kv} L_k} B_v. \tag{2.11}$$

Next, we define r_q' as

$$r_q' = \frac{L_q}{\sum_{k=1}^{N} A_{ku} L_k} B_u. \tag{2.12}$$

Based on Equations (2.11) and (2.12), we have

$$r_q' > r_q^* \tag{2.13}$$

$$et_q' = \frac{L_q}{r_q'} < \frac{L_q}{r_q^*} = et_q^*. \tag{2.14}$$

On the other hand, putting Equations (2.9) and (2.12) together, we have

$$et_p^* = \frac{L_p}{r_p^*} = \frac{\sum_{k=1}^{N} A_{ku} L_k}{B_u} = \frac{L_q}{r'_q} = et'_q. \tag{2.15}$$

Now using the Inequity (2.14) and Equation (2.15), we get

$$et_p^* = et'_q < et_q^*, \tag{2.16}$$

which conflicts with the assumption that $et_p^* = \max_{i=1,\ldots,N}\{et_i^*\}$. Therefore, there does not exist a variable r_q^* and a constraint v that satisfies the Equation (2.11). As a result, we have proved the lemma. □

Based on Lemma 1, we have the following theorem.

Theorem 1. *The vector $\vec{r}^*$ defined in Equation 2.8 is an optimal solution of the OptRA problem.*

Proof. Assume that $et_p^* = L_p/r_p^* = \max_{i=1,\ldots,N}\{et_i^*\}$ and r_p^* obtains the minimum value when constraint u is considered. Then, according to Lemma 1, there exists

$$r_i^* = \frac{L_i}{\sum_{k=1}^{N} A_{ku} L_k} B_u, \ \forall i \ that \ A_{iu} \neq 0. \tag{2.17}$$

Based on this equation, for every i with $A_{iu} \neq 0$, we have

$$et_i^* = \frac{L_i}{r_i^*} = \frac{\sum_{k=1}^{N} A_{ku} L_k}{B_u} = \frac{L_p}{r_p^*} = et_p^*. \tag{2.18}$$

Now assume that instead of $\vec{r}^*$, a vector $\vec{r}\prime$ is the optimal solution of the

problem. Also assume that $et'_q = L_q/r'_q = \max_{i=1,\ldots,N}\{et'_i\}$. Then, we have

$$et'_q < et^*_p. \tag{2.19}$$

Using Equation (2.18), Equation (2.19) and the assumption that $et'_q = \max_{i=1,\ldots,N}\{et'_i\}$, we can obtain

$$et'_i \leq et'_q < et^*_p = et^*_i,\ \forall i\ that\ A_{iu} \neq 0. \tag{2.20}$$

Intuitively, we then have

$$r'_i > r^*_i,\ \forall i\ that\ A_{iu} \neq 0. \tag{2.21}$$

On the other hand, from Equation (2.17), we can get

$$\sum_{i=1}^{N} A_{iu} r^*_i = B_u. \tag{2.22}$$

Putting Inequity (2.21) and Equation (2.22) together, we have

$$\sum_{i=1}^{N} A_{iu} r'_i > B_u \tag{2.23}$$

which conflicts with the assumption that $\vec{r}'$ is a feasible solution. As a result, there does not exist a feasible solution that is better than $\vec{r}^*$. Therefore, the vector $\vec{r}^*$ defined in Equation (2.8) is an optimal solution of the OptRA problem. □

We develop an algorithm, named Optimal Resource Allocation (ORAl-

Algorithm 1 Optimal Resource Allocation (ORAlloc)

```
1: function ORAlloc(A_ij, B_j, L_i)
2:   for i ← 1 to N do   r_i^* ← ∞
3:   for j ← 1 to R do
4:     for i ← 1 to N do   L_sum ← L_sum + A_ij L_i
5:     for i ← 1 to N do
6:       if (L_i / L_sum) B_j ≤ r_i^* then r_i^* ← (L_i / L_sum) B_j
7:     end for
8:   end for
9:   for i ← 1 to N do   et_i^* ← L_i / r_i^*   end for
10:  return {r_i^*, et_i^*}
11: end function
```

loc), to calculate the optimal solution (2.8). The algorithm iterates through every task t_i; for each task, it calculates the term $\frac{L_i}{\sum_{k=1}^{N} A_{kj} L_k} B_j$ for every constraint and assign the minimum value among all terms to r_i^*. Pseudocode of ORAlloc is shown in Algorithm 1. Straightforwardly, the runtime complexity of ORAlloc is $O\big(N \cdot R\big)$.

2.3 The OptRA-Total Problem

Based on the objective and constraints introduced in Section 2.1, we can present the OptRA-Total problem as

OptRA-Total

Variables

- r_{t_i}: resources allocated to task t_i.
- et_{t_i}: the execution time of task t_i.

Constants

- N: the number of tasks in the set $\mathcal{T}$.
- K: the number of constraints in the problem.
- $Load_i$: the load of task t_i.
- A_{ik}: coefficient of variable r_{t_i} in the kth constraint.
- B_k: the total resources allowed to be allocated in the kth constraint.

Objective

$$Minimize \quad \sum_{i=1}^{N} \frac{Load_i}{r_{t_i}}, \tag{2.24}$$

Subject to

$$\sum_{i=1}^{N} A_{ik} r_{t_i} \leq B_k, \ k = 1, \ldots, K, \tag{2.25}$$

$$r_{t_i} \geq 0, \ i = 1, \ldots, N. \tag{2.26}$$

We next show that the OptRA-Total problem is a convex optimization problem. We observe that the function $Load_i/r_{t_i}$ is convex, because its second derivative is nondecreasing when r_{t_i} is larger than 0. Therefore, according to [31], the objective function, as the sum of a convex function, is also a convex function. In addition, the functions in constraints (2.25) and (2.26) are all affine on r_{t_i} and thus convex. As a result, the OptRA-Total problem is a convex optimization problem which can be efficient solved by using convex

algorithms introduced in [31].

Chapter 3

Energy-aware Scheduling of Embarrassingly Parallel Jobs and Resource Allocation in Cloud

3.1 Introduction

In recent years, we have witnessed a dramatic increasing use of cloud computing techniques as it enables on-demand provisioning of computing resources and platforms for users [32]. In a cloud system, users can easily access the required computing resources, while the underlying infrastructure (i.e., data centers composed of physical servers,) is hidden from them and user jobs are executed on Virtual Machines (VMs) whose location is unknown from these users. By deploying the applications or executing the jobs in a cloud, cloud

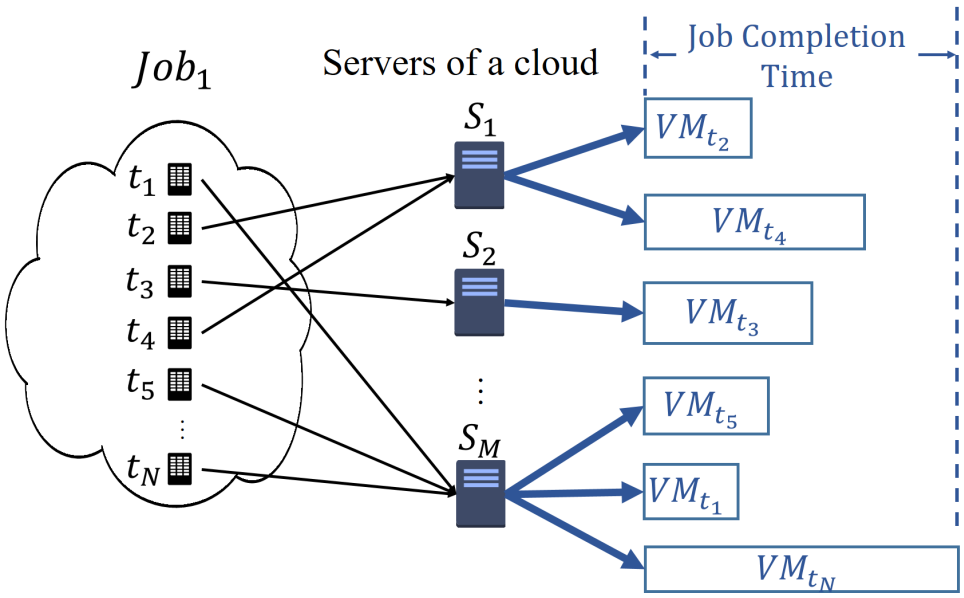


Figure 3.1: Job Execution Model.

users are able to avoid the cost and responsibility of purchasing, setting up, and maintaining the hardware and software infrastructures and thereby focus more on their missions [33].

In contrast to cloud users' unawareness of the infrastructures hidden behind the cloud, cloud providers have full control of the infrastructure. By widely using modern virtualization techniques, cloud providers can flexibly place jobs on suitable physical servers and dynamically allocate computing resources to user jobs in the form of VMs while keeping the provisioned VMs isolated and interference free from each other. As a large number of user jobs can be simultaneously executed in a cloud, one of the cloud provider's important responsibilities is to properly schedule these jobs and determine an appropriate sharing of resources among these user jobs.

As a cloud provider, scheduling user jobs in a way that minimizes their completion time is very important: With smaller job completion times, the cloud can execute more user jobs; For public clouds, this means generating more profit, while for private clouds, this means higher throughout and therefore usually higher productivity. However, scheduling jobs while minimizing their completion time can be a challenging problem in clouds.

In this paper, we study the job scheduling problem and focus on scheduling embarrassingly parallel jobs which are composed of a set of independent tasks with very minimal or no data synchronization. A large number of applications belong to this type of jobs. Examples include distributed relational database queries, Monte Carlo simulations, BLAST searches, parametric studies, and image processing applications such as ray tracing [34]. To execute an embarrassingly parallel job, each of its tasks is placed on a physical server and executed in a VM created for that task. The completion time of this job is the completion time of the last finished task, i.e., the makespan of that set of tasks. Fig. 3.1 shows the execution model of an embarrassingly parallel job. Scheduling such a job includes determining a task placement plan that indicates the servers to execute each task in the job and a resource allocation plan that indicates the amount of computing resources allocated to each task.

To schedule embarrassingly parallel jobs with the goal of minimizing the Job Completion Time (JCT), we need to answer three questions:

1. How to optimally allocate computing resources to a job with pre-determined task placement plan?

2. How to place tasks and allocate resources for one job?

3. How to address multiple jobs in online scheduling?

Naturally, these three questions are in a progressive relationship. By answering the previous question, we essentially reduce the complexity of the later question. While several approaches has been proposed to schedule independent tasks in data centers [35–42], none of them consider task placement and resource allocation together. Motivated by this, in this chapter, we focus on the problem of scheduling embarrassingly parallel jobs and propose solutions to the three questions shown above.

Moreover, we consider the energy consumption for executing a job during the scheduling procedure. Along with the rapidly increasing number of cloud users, more and more large-scale data centers comprising tens of thousands of servers are built recently, which leads tremendous amount of energy consumption with huge cost [43]. High energy consumption also reduces system reliability and has negative impacts on the environment [44]. Consequently, reducing the total energy consumption of a cloud is highly desirable. While some approaches [45–47] with the objective of reducing the total energy consumption of a cloud have been proposed, in this chapter, we focus on scheduling jobs with the goal of minimizing their completion time but constrain the total energy consumed for executing a job.

In summary, our main contributions include

- We formally define the problem of scheduling embarrassingly parallel jobs. We derive a job energy consumption model based on the existing VM power model proposed by other researchers and then formulate the energy consumption constraint. We also formulate the resource availability constraints which limit the amount of resources that can be used by a job.

(Section 3.3)

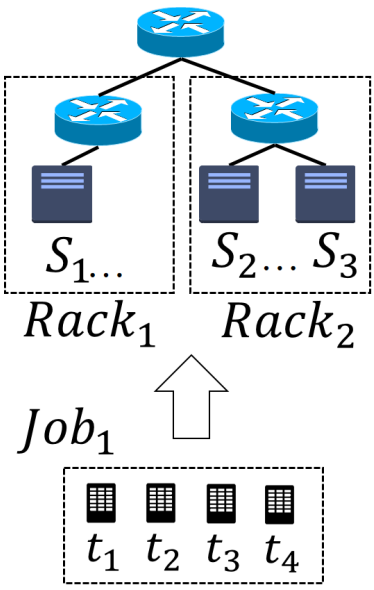
- We formulate the problem of optimal resource allocation with pre-determined task placement (OptRA) as a convex optimization problem and present an analytical solution of this problem. (Section 3.4.1)

- We study the problem of scheduling a single embarrassingly parallel job (SJS) and formulate it as a Non-linear Mixed Integer Programming (NLMIP) problem. We propose a relaxation in which the tasks are assumed to be divisible and transform it to a Linear Programming (LP) problem. (Section 3.4.2)

- We propose an algorithm named Task Placement and Resource Allocation (TaPRA) and its simplified version TaPRA-fast that solve the SJS problem based on the solution of the relaxed problem. (Section 3.4.3)

- We propose an online scheduler named OnTaPRA to address multiple jobs in online scheduling. The OnTaPRA scheduler periodically schedules all jobs in the waiting queue by using Shortest Job First (SJF) scheduling policy. For work conservation, it distributes residual capacity of servers to running tasks. (Section 3.5)

- We evaluate the performance of the proposed TaPRA algorithm and OnTaPRA scheduler through simulations. In offline simulations, we compare the TaPRA algorithm with some existing algorithms. The simulations results show that the TaPRA and TaPRA-fast algorithm can achieve 40%-430% smaller JCT than the existing algorithms. In online simulations, we compare the OnTaPRA scheduler with some existing schedulers. The sim-



Schedule A with JCT of 33.33

ID	Load	Placed host	Allocated resource	Execution time	JCT
t_1	100	S_2	3	30	33.33
t_2	200	S_1	30	6.67	
t_3	300	S_3	12	33.33	
t_3	400	S_3	15	33.33	

Schedule B with JCT of 20

ID	Load	Placed host	Allocated resource	Execution time	JCT
t_1	100	S_1	5	20	20
t_2	200	S_2	10	20	
t_3	300	S_3	20	20	
t_3	400	S_1	25	20	

Figure 3.2: Example of scheduling a embarrassingly parallel job in cloud.

ulations results show that the OnTaPRA scheduler can achieve 60%-280% smaller average JCT that the existing schedulers. In addition, the results also show that TaPRA-fast can be 10 times faster than TaPRA with around 5% performance degradation compared to TaPRA, which makes the use of TaPRA-fast very applicable in practice (Section 3.6)

3.1.1 An Example

To illustrate the job scheduling problem, consider the example shown in Fig. 3.2, in which we need to schedule an embarrassingly parallel job composed of four tasks (t_1, t_2, t_3, t_4) onto three servers (s_1, s_2, s_3) in a data center. Let the available computing resources (for example, the number of

CPU cores) on three servers be (35, 10, 30) and load of the four tasks be (100, 200, 400, 500). [1] We further consider a resource availability constraint: the total computing resources allocated to this task set in each rack cannot exceed 30. Fig. 3.2 also shows two schedules: Schedule A with JCT of 33.33 and Schedule B with JCT of 20. Schedule B has smaller JCT because it proportionally allocates computing resources to tasks. In this way, the maximum task execution time, i.e., the JCT, is efficiently reduced. From this example, we can see that task placement plan and resource allocation plan together determine the JCT. We can achieve the minimum JCT only if we find out the optimal solution on both of them.

However, when the problem scale is large in practice, there exists a vast amount of possible task placement plans and for each placement plan there are very many ways to allocate resources. Searching for the optimal solution in such a huge solution space is not an easy problem to solve. This problem becomes even more complex, when we consider the energy consumption limitation.

3.2 Related Work

A significant amount of research has focused on task/job scheduling and resource allocation in clouds. In this section, we discuss some of the research works that we consider most relevant to our problem from the following aspects: (1) Approaches focusing on scheduling performance, like

[1] We temporarily omit the meaning of quantified computing power and task load and simply assume that the execution time of a task is inversely proportional to the amount of resources allocated to the task.

response time, makespan, and completion time; (2) Energy-aware scheduling approaches which set the scheduling goal as minimizing energy consumption or consider the energy consumption during scheduling; (3) Online scheduling approaches which focus on proposing an online scheduler or scheduling policy.

Approaches Focusing on Scheduling Performance. This type of approaches mainly focus on optimizing the time-related performance, like response time, makespan or completion time [35–42]. Comprehensive surveys about task scheduling and resources scheduling in this category can be found in [35,36]. Zuo *et al.* [37] propose a multi-objective Ant Colony Algorithm to solve the task scheduling problem. This algorithm considers the makespan and the user's budget costs as constraints of the optimization problem. Tang *et al.* [39] propose a self-adaptive scheduling algorithm for MapReduce jobs. The algorithm decides the start time point of each reduce task dynamically according to each job context, including the task completion time and the size of map output. Tsai *et al.* [40] propose a hyper-heuristic scheduling algorithm which dynamically determines which low-level heuristic is to be used in find better candidate solutions for scheduling tasks in cloud. Verma *et al.* [41] propose an improved Genetic Algorithm which uses the outputs of Max-Min and Min-Min as initial solutions to scheduling independent tasks. Gan *et al.* [42] propose a Genetic Simulated Annealing algorithm to optimize the makespan of a set of tasks, in which Simulated Annealing is used to optimize each offspring generated by the Genetic algorithm. However, all these proposed approaches consider the computing resources allocated to each task as static. Without benefiting from dynamic resource allocation, the

efficiency of these approaches in solving the task scheduling problem may be diminished.

Energy-aware Task Scheduling. Energy-aware task scheduling has also been given great attention [22, 23, 47–50]. Shen *et al.* [48] propose a genetic algorithm to achieve adaptive regulations for different requirements of energy and performance in cloud tasks. In this algorithm, two fitness functions for energy and task completion time are designed for optimizations. Zhao *et al.* [49] propose an energy and deadline aware task scheduling method which models the data-intensive tasks as binary trees.The proposed method aims to schedule Directed Acyclic Graph (DAG)-like workflows. In contrast, in this chapter, we focus on embarrassingly parallel jobs composed of independent tasks. Hosseinimotlagh *et al.* [23] propose a VM scheduling algorithm that allocates resources to VMs in a way that the optimal energy level of the host of those VMs is reached. The proposed algorithm assumes that the VMs are pre-mapped onto a host and focuses on allocating resources to the VMs. In contrast to this algorithm, our algorithms determine the task placement. Wu *et al.* [22] develop a scheduling algorithm for the cloud datacenter with a DVFS technique. The algorithm schedules one job at a time and does not consider about co-scheduling between jobs. In addition, this algorithm pre-defines several frequency ranges with corresponding voltage supply and determines a specific range for the job. Mhedheb *et al.* [47] propose a thermal-aware VM scheduling mechanism that achieves both load balance and temperature balance with the final goal of reducing energy consumption. Mhedhed *et al.* analyze the impact of VM migration on energy consumption and utilize the VM migration technique in the proposed mech-

anism to lower the host temperature. Xiao *et al.* [50] propose a system that dynamically combine VMs running different types of workloads together to improve the overall utilization of resources and reduce the number of running servers, which reduces the energy consumption. The goal of these introduced energy-aware approaches is to minimize the energy consumption in clouds. In contrast, in this chapter, we consider the energy consumption as a constraint and set our goal as minimizing the job completion time.

Online Scheduling. How to address multiple tasks/jobs in online scheduling is also an important question which has attracted a lot attention [44, 51–54] Shin *et al.* [51] modify the conservative backfilling algorithm by utilizing the earliest deadline first and largest weight first policies to address the waiting jobs according to their deadline. The objective of this algorithm is to guarantee the job deadline while improving resource utilization, which is different from the our objective in this paper. Zhu *et al.* [44] design a rolling-horizon scheduling architecture for real-time task scheduling in clouds, which includes an energy consumption model and an energy-aware scheduling algorithm. However, in the proposed architecture, the tasks are scheduled separately. In contrast, in this chapter, we addresses tasks belonging to one job together to minimize the job compltion time. Liu *et al.* [52] propose an online scheduler that allows VMs to obtain extra CPU shares when blocked by I/O interrupted recently and thereby reduces the energy-efficiency losses caused by I/O-intensive tasks. Ge *et al.* [54] propose an GA-based task scheduler which evaluates all the waiting tasks and uses a genetic algorithm to schedule these tasks with the goal of achieving better load balance.

3.3 Problem Definition

In this paper, we consider scheduling jobs composed of independent tasks in a data center comprising heterogeneous servers. To schedule such a job, we are required to place each of its tasks onto a server and launch a VM with certain amount of computing capacity to execute that task. We now introduce input, output, objective function and constraints considered in the problem.

3.3.1 Input

The input contains user jobs submitted by users at different times and a data center with a set of heterogeneous servers used to execute those jobs.

Job. A job J comprise a set of independent tasks. It is defined as $J = \{T_1, T_2, \ldots, T_N\}$ in which T_i is the ith independent task of that job. The load $Load_i$ of the task T_i is defined as the execution time of T_i, when it is placed on a unit-efficiency server and executed on a VM with unit computing capacity.

Data Center. The data center used to execute user jobs is defined as $DC = \{S_1, S_2, \ldots, S_M\}$ in which S_j is the jth server in the data center. The available computing resources of server S_j is denoted by C_{s_j}.

Due to the heterogeneity of servers, different severs may have different efficiencies of executing the same task [12]. To model such heterogeneity, we denote the efficiency of executing task T_i on server S_j by λ_{ij} ($\lambda_{ij} \in (0, 1]$). Correspondingly, when a task T_i is placed on a server S_j with efficiency λ_{ij} and executed on a VM with computing power c_{t_i}, the execution time of T_i,

denoted by et_{t_i}, is

$$et_{t_i} = \frac{Load_i}{\lambda_{ij} c_{t_i}}. \tag{3.1}$$

3.3.2 Output

For each input job, the output contains a task placement plan and a resource allocation plan.

Task placement plan. A task placement plan indicates the servers to execute the input job's tasks. We use binary variables x_{ij} to present such a plan. Specifically, if task T_i is placed on server S_j, the value of x_{ij} is 1; otherwise its value is 0. We can express it as

$$x_{ij} = \begin{cases} 1, & \text{if } T_i \text{ is placed on } S_j, \\ 0, & \text{otherwise.} \end{cases} \tag{3.2}$$

Resource allocation plan. A resource allocation plan indicates the amount of computing resources allocated to each task, i.e., the computing capacity allocated to the VM created to execute the tasks. We denote the set of VMs by $\mathcal{VM} = \{VM_{t_1}, \ldots, VM_{t_M}\}$, where VM_{t_i} is the VM created to execute task T_i, and denote the resource allocation plan by $\vec{c} = \{c_{t_1}, \ldots, c_{t_N}\}$ in which c_{t_i} is the amount of computing resources allocated to VM_{t_i}.

3.3.3 Objective

Single job scheduling. When scheduling a single job, our objective is to determine a task placement plan and a resource allocation plan while minimizing the job completion time (JCT). Because a job is not completed

until all its tasks finish, the JCT essentially equals the completion time of the last finished task. Let the completion time of task T_i be et_{t_i}, based on Equation (3.1), the objective function is

$$Minimize \quad \max_{i=1,\ldots,N} \left\{ et_{t_i} \mid et_{t_i} = \frac{Load_i}{\sum_{j=1}^{M} x_{ij}\lambda_{ij}c_{t_i}} \right\}. \tag{3.3}$$

Online scheduling. In online scheduling, multiple jobs arrive in a time sequence and we are required to schedule all these jobs. In this situation, our objective is to minimize the average completion time of these jobs, i.e., the average JCT.

3.3.4 Constraints

Task placement constraints. Because each task should be placed on only one server, we have the following constraint:

$$\sum_{j=1}^{M} x_{ij} = 1, \ i = 1, \ldots, N. \tag{3.4}$$

Resource availability constraints. When allocating resources to the tasks of a job, there may be limitations on the total amount of computing resource that can be allocated to that job on a server or a subset of servers. Such constraints are usually enforced by system administrator to maintain proper sharing of resources among multiple jobs belong to different users. We can

Table 3.1: List of constants and variables.

Constants	
Name	Description
J	the input embarrassingly parallel job
T_i	the ith independent task of job J
$Load_i$	the load of the task T_i
S_j	the jth server in the data center
C_{s_j}	the available computing resource on server S_j
VM_{t_i}	the VM executing task T_i
$\alpha_{VM_{t_i}}$	the power model constant of VM_{t_i}
λ_{ij}	the efficiency of executing task T_i on server S_j
A_{jk}	the coefficient in the kth availability constraint for S_j
B_k	the total allowed resources in the kth availability constraint
X_{ij}	the determined placement between task T_i and server S_j
E_{MAX}	the maximum allowed energy consumption
Variables	
Name	Description
x_{ij}	the placement relationship between task T_i and server S_j
c_{t_i}	the amount of computing resources allocated to T_i
c_{s_j}	the total amount of resources allocated to the job on S_j
et_{t_i}	the completion time of task T_i
l_{ij}	the load of sub-task T_{ij}
$c_{t_{ij}}$	the amount of resources allocated to sub-task T_{ij}

formulate these constraints as

$$\sum_{j=1}^{M} A_{jk} \sum_{i=1}^{N} x_{ij} c_{t_i} \leq B_k,\ k = 1, \ldots, R. \tag{3.5}$$

where $\sum_{i=1}^{N} x_{ij} c_{t_i}$ is the total computing capacity allocated to the input jobs on server S_j; A_{jk} is the coefficient in the kth constraint; B_k is the total resources allowed to be allocated in the kth constraint.

Energy consumption constraints. As the energy cost can contribute a significant part of operating cost of a data center [43], the system administrator may also limit the total amount of energy that can be consumed by a job. To formulate this energy consumption constraint, we first consider the VM power model and energy consumption model.

VM Power Model. Power modeling for VMs in data centers has attracted significant attention [55–58]. While both linear and non-linear power model are proposed, the linear model is the most widely used method in the estimation of power consumption [58], whose accuracy has been proved [55–57]. The linear VM power model have also been used in many existing task scheduling and resource allocation approaches [33, 44, 48, 59].

In the linear model, the total power consumption P_s of a physical server is composed of the static power P_{static} and the dynamic power $P_{dynamic}$. While P_{static} is usually constant regardless of whether VMs are running or not, as long as the server is turned on, $P_{dynamic}$ is consumed by VMs running on the server. Suppose that n VMs are running on a server, then the server power

consumption P_s is

$$P_s = P_{static} + \sum_{i=1}^{n} P_{VM_i}, \text{subject to} \sum_{i=1}^{n} c_{VM_i} \leq C_s, \tag{3.6}$$

in which C_s is the total computing capacity of the server; P_{VM_i} and c_{VM_i} are the power consumption and the computing capacity of VM i. P_{VM_i} can be further decomposed into power of components such as CPU, memory, disk and IO devices [56], thus it can be calculated as

$$P_{VM_i} = P_{VM_i}^{CPU} + P_{VM_i}^{Memory} + P_{VM_i}^{Disk} + P_{VM_i}^{IO}. \tag{3.7}$$

In this paper, we mainly focus on the power consumption of CPU utilized by a VM, because the CPU utilization of a VM is directly related to the execution time of tasks running on that VM. Therefore, we approximate the VM power consumption by the CPU power consumption of a VM, following similar setting in existing work [33]. A utilization based VM power model then is

$$P_{VM_i} = P_{VM_i}^{CPU} = \alpha_{VM_i} \cdot c_{VM_i}, \tag{3.8}$$

where P_{VM_i} is the power consumption of VM_i, c_{VM_i} is the amount of computing resources allocated to the VM and α_{VM_i} is model specific constant, following similar model proposed in previous work [56, 58].

Based on the VM power model (3.8), the energy consumed by VM VM_{t_i}, denoted by $E_{VM_{t_i}}$, is

$$E_{VM_{t_i}} = P_{VM_{t_i}} \cdot et_{t_i} = \alpha_{VM_{t_i}} c_{t_i} \cdot et_{t_i} = \frac{\alpha_{VM_{t_i}} Load_i}{\sum_{j=1}^{M} x_{ij} \lambda_{ij}}. \tag{3.9}$$

Let E_{total} denote the total energy consumption of the input job and E_{MAX} denote the maximum energy consumption allowed, the energy consumption constraint is

$$E_{total} = \sum_{t=1}^{N} E_{VM_{t_i}} = \sum_{i=1}^{N} \frac{\alpha_{VM_{t_i}} Load_i}{\sum_{j=1}^{M} x_{ij}\lambda_{ij}} <= E_{MAX}. \tag{3.10}$$

3.4 Scheduling a Single Job with Independent Tasks

In this section, we focus on scheduling a single job. We start from its subproblem: Optimal Resource Allocation with Pre-determined Task Placement (OptRA).

3.4.1 Optimal Resource Allocation with Pre-determined Task Placement (OptRA)

In the OptRA problem, the task placement plan has already been determined, i.e., the value of x_{ij} is known. For convenience and clarity, we use X_{ij} to indicate the determined task placement plan. Our goal is to allocate computing resources to these tasks while minimizing the JCT.

With the determined task placement, the objective (3.3) becomes

$$Minimize \max_{i=1,\ldots,N} \left\{ et_{t_i} \mid et_{t_i} = \frac{\frac{Load_i}{\sum_{j=1}^{M} X_{ij}\lambda_{ij}}}{c_{t_i}} \right\}. \tag{3.11}$$

The resource availability constraints (3.5) become

$$\sum_{i=1}^{N}\sum_{j=1}^{M} X_{ij}A_{jk}c_{t_i} \leq B_k,\ k = 1, \ldots, R. \tag{3.12}$$

We can then formulate the OptRA problem as

OptRA

$$Minimize \max_{i=1,\ldots,N} \left\{ et_{t_i} \mid et_{t_i} = \frac{L_{t_i}}{c_{t_i}} \right\}, \tag{3.13}$$

Subject to

$$\sum_{i=1}^{N} P_{ik}c_{t_i} \leq B_k, k = 1, \ldots, R, \tag{3.14}$$

$$c_{t_i} \geq 0,\ i = 1, \ldots, N, \tag{3.15}$$

$$L_{t_i} = \frac{Load_i}{\sum_{j=1}^{M} X_{ij}\lambda_{ij}}, P_{ik} = \sum_{j=1}^{M} X_{ij}A_{jk}. \tag{3.16}$$

Remarks:

- While L_{t_i} and P_{ik} are used to simplify the formulation of the problem, L_{t_i} also stands for the equivalent load of T_i considering the execution efficiency of the server on which T_i is placed; P_{ik} is the coefficient of T_i in the kth availability constraint.

- Task placement constraints (3.4) and energy consumption constraints (3.10) are not included in OptRA, as they are only related with the variable x_{ij} whose value is already determined in the above problem.

We observe that the constraints (3.14) and (3.15) are all affine on c_{t_i}. Meanwhile, the function et_{t_i} is convex, because its second derivative is nondecreasing when c_i is larger than 0. Therefore, according to [31], the objective function, i.e., the pointwise maximum function of et_{t_i}, is also a convex function. As a result, OptRA is a convex optimization problem.

3.4.1.1 Analytical Solution

While existing convex optimization algorithms [31] can be used to solve the OptRA problem, we develop an analytical solution which is more efficient. Specifically, we define a vector $\vec{c}^* = \{c_1^*, c_2^*, \ldots, c_N^*\}$ with

$$c_i^* = \min_{k \in \mathcal{R}_i} \left\{ \frac{L_{t_i}}{\sum_{j=1}^{N} P_{jk} L_{t_j}} B_k \right\}, \ i = 1, \ldots, N, \tag{3.17}$$

where $\mathcal{R}_i$ is the set of constraints in which the coefficient of variable c_i is not zero. We now show that this vector $\vec{c}^*$ is an optimal solution of the OptRA problem. We have the following lemma and theorem.

Lemma 2. *Assume that for the vector $\vec{c}^*$ defined in Equation 3.17, task p has the largest finish time, i.e., $et_p^* = L_{t_p}/c_p^* = \max_{i=1,\ldots,N}\{et_i^*\}$. Also assume that c_p^* obtains the minimum value when constraint u is considered,*

$$c_p^* = \min_{k \in \mathcal{R}_p} \left\{ \frac{L_{t_p}}{\sum_{j=1}^{N} P_{jk} L_{t_j}} B_k \right\} = \frac{L_{t_p}}{\sum_{j=1}^{N} P_{ju} L_{t_j}} B_u. \tag{3.18}$$

then every c_i subjected to constraint u, i.e., $P_{iu} \neq 0$, obtains the minimum

value when constraint u is considered, i.e.,

$$c_i^* = \min_{k \in \mathcal{R}_i} \left\{ \frac{L_{t_i}}{\sum_{j=1}^{N} P_{jk} L_{t_j}} B_k \right\} = \frac{L_{t_i}}{\sum_{j=1}^{N} P_{ju} L_{t_j}} B_u. \tag{3.19}$$

Proof. To begin with, assume that there exists a variable c_q^* with $P_{qu} \neq 0$, which gets the minimum value when constraint v (other than constraint u) is considered, i.e.,

$$c_q^* = \min_{k \in \mathcal{R}_q} \left\{ \frac{L_{t_q}}{\sum_{j=1}^{N} P_{jk} L_{t_j}} B_k \right\} = \frac{L_{t_q}}{\sum_{j=1}^{N} P_{jv} L_{t_j}} B_v. \tag{3.20}$$

Next, we define c_q' as

$$c_q' = \frac{L_{t_q}}{\sum_{j=1}^{N} P_{ju} L_{t_j}} B_u. \tag{3.21}$$

Based on Equations (3.20) and (3.21), we have

$$et_q' = \frac{L_{t_q}}{c_q'} < \frac{L_{t_q}}{c_q^*} = et_q^*. \tag{3.22}$$

On the other hand, based on Equation (3.18), we have

$$et_p^* = \frac{L_{t_p}}{c_p^*} = \frac{\sum_{j=1}^{N} P_{ju} L_j}{B_u} = \frac{L_{t_q}}{c_q'} = et_q'. \tag{3.23}$$

Now using the Inequality (3.22) and Equation (3.23), we get

$$et_p^* = et_q' < et_q^*, \tag{3.24}$$

which conflicts with the assumption that $et_p^* = \max_{i=1,\ldots,N}\{et_i^*\}$. Therefore, there does not exist a variable c_q^* and a constraint v that satisfies the Equation (3.20). As a result, we have proved the lemma. □

Theorem 2. *The vector $\vec{c}^*$ defined in Equation 3.17 is an optimal solution of the OptRA problem.*

Proof. Assume that $et_p^* = L_{t_p}/c_p^* = \max_{i=1,\ldots,N}\{et_i^*\}$ and c_p^* obtains the minimum value when constraint u is considered. Then, according to Lemma 2, there exists

$$c_i^* = \frac{L_{t_i}}{\sum_{j=1}^{N} P_{ju} L_j} B_u, \ \forall i \ that \ P_{ju} \neq 0. \tag{3.25}$$

Based on this equation, for every i with $P_{iu} \neq 0$, we have

$$et_i^* = \frac{L_{t_i}}{c_i^*} = \frac{\sum_{j=1}^{N} P_{ju} L_{t_j}}{B_u} = \frac{L_{t_p}}{c_p^*} = et_p^*. \tag{3.26}$$

Now assume that instead of $\vec{c}^*$, a vector $\vec{c}\prime$ is the optimal solution of the problem. Also assume that $et_q' = L_{t_q}/c_q' = \max_{i=1,\ldots,N}\{et_i'\}$. Together with Equation (3.26), we have

$$et_i' \leq et_q' < et_p^* = et_i^*, \ \forall i \ that \ P_{iu} \neq 0. \tag{3.27}$$

Naturally, we have

$$c_i' > c_i^*, \ \forall i \ that \ P_{iu} \neq 0. \tag{3.28}$$

On the other hand, from Equation (3.25), we can get

$$\sum_{i=1}^{N} P_{iu} c_i^* = B_u. \tag{3.29}$$

Putting Inequity (3.28) and Equation (3.29) together, we have

$$\sum_{i=1}^{N} P_{iu} c_i' > B_u, \tag{3.30}$$

which conflicts with the assumption that $\vec{c}'$ is a feasible solution. As a result, there does not exist a feasible solution that is better than $\vec{c}^*$. Therefore, the vector $\vec{c}^*$ defined in Equation (3.17) is an optimal solution of the OptRA problem. □

Note that the analytical solution (3.17) has important usage when scheduling a single job. It essentially reduces the dimension of the problem: For any determined task placement plan, we can use Equation (3.17) to calculate the corresponding optimal resource allocation plan.

3.4.2 Formulation of the Single Job Scheduling Problem and Its Solvable Relaxation

We now study the Single Job Scheduling (SJS) problem in which we are required to determine the task placement plan and resource allocation plan for an input job. The task execution model of the input job has been shown in Fig. 3.1.

Based on the objective and constraints introduced in Section 3.3, the SJS

problem can be formulated as

SJS

$$Minimize \quad \max_{i=1,\ldots,N} \left\{ et_{t_i} \mid et_{t_i} = \frac{Load_i}{\sum_{j=1}^{M} x_{ij}\lambda_{ij}c_{t_i}} \right\}, \tag{3.31}$$

Subject to

$$\sum_{i=1}^{N} \frac{\alpha_{VM_{t_i}} Load_i}{\sum_{j=1}^{M} x_{ij}\lambda_{ij}} \leq E_{MAX}, \tag{3.32}$$

$$\sum_{j=1}^{M} A_{jk} \sum_{i=1}^{N} x_{ij}c_{t_i} \leq B_k, \ k = 1, \ldots, R, \tag{3.33}$$

$$\sum_{j=1}^{M} x_{ij} = 1, \ i = 1, \ldots, N, \tag{3.34}$$

$$x_{ij} = 0 \ or \ 1, \ i = 1 \ldots, N, \ j = 1, \ldots, M, \tag{3.35}$$

$$c_{t_i} \geq 0, \ i = 1, \ldots, N. \tag{3.36}$$

Remarks:

- The objective (3.31) and constraints (3.32)-(3.34) are formally defined in Section 3.3.
- Constraints (3.35) and (3.36) are domain constraints.

Naturally, the SJS problem is a NLMIP problem which is hard to solve directly. To solve this problem, we first propose a solvable relaxation and

then determine a solution of the SJS problem based on the solution of the relaxation.

3.4.2.1 A Relaxation of the SJS problem and An Equivalent Linear Programming Problem

Because the SJS problem is a NLMIP problem, a straightforward relaxation is relaxing the binary variable x_{ij} to a real variable. However, due to the term $x_{ij}c_{t_i}$, this relaxation is a NLP problem which is still hard to solve.

To obtain a solvable relaxation, we assume that the tasks of the input job are divisible [60–62] and each task T_i is divided into M sub-tasks and placed on M servers respectively. Let t_{ij} denote the sub-task of T_i placed on server S_j and let l_{ij} denote the load of t_{ij}. A VM is then created for each sub-task placed on each server. Fig. 3.3 shows this execution model.

Furthermore, let $VM_{t_{ij}}$ denote the VM created by server S_j to execute sub-task t_{ij} and let $c_{t_{ij}}$ denote the amount of resources allocated to $VM_{t_{ij}}$. The SJS problem is now relaxed to a problem of determining the value of l_{ij} and $c_{t_{ij}}$ with the goal of minimizing the JCT. We name this new problem as **SJS-Relax-Divisible**.

The execution time of sub-task t_{ij}, denoted by $et_{t_{ij}}$, is

$$et_{t_{ij}} = \frac{l_{ij}}{\lambda_{ij} c_{t_{ij}}}. \tag{3.37}$$

The energy consumed for executing t_{ij} is

$$E_{t_{ij}} = P_{VM_{t_{ij}}} \cdot et_{t_{ij}} = \alpha_{VM_{t_{ij}}} c_{t_{ij}} \cdot et_{t_{ij}} = \alpha_{VM_{t_{ij}}} \cdot \frac{l_{ij}}{\lambda_{ij}}. \tag{3.38}$$

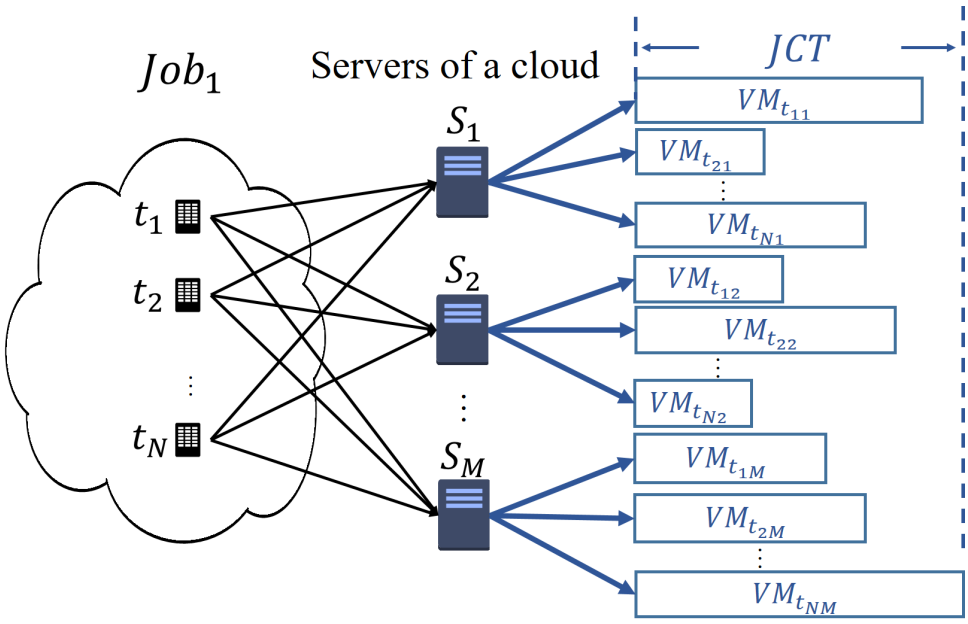


Figure 3.3: Job Execution Model for SJS-Relax-Divisible.

The total resources allocated by server S_j, i.e., c_{s_j}, is

$$c_{s_j} = \sum_{i=1}^{N} c_{t_{ij}}. \tag{3.39}$$

Based on the above equations, the SJS-Relax-Divisible problem can be formulated as

SJS-Relax-Divisible

$$Minimize \quad \max_{\substack{i=1,\ldots,N \\ j=1,\ldots,M}} \left\{ et_{t_{ij}} \mid et_{t_{ij}} = \frac{l_{ij}}{\lambda_{ij} c_{t_{ij}}} \right\}, \tag{3.40}$$

Subject to

$$\sum_{i=1}^{N}\sum_{j=1}^{M}\alpha_{VM_{t_{ij}}}\cdot\frac{l_{ij}}{\lambda_{ij}}\leq E_{MAX}, \tag{3.41}$$

$$\sum_{j=1}^{M}A_{jk}\sum_{i=1}^{N}c_{t_{ij}}\leq B_k,\ k=1,\ldots,R, \tag{3.42}$$

$$\sum_{j=1}^{M}l_{ij}=Load_i,\ i=1,\ldots,N, \tag{3.43}$$

$$c_{t_{ij}}\geq 0,\ l_{ij}\geq 0,\ i=1,\ldots,N,\quad j=1,\ldots,M. \tag{3.44}$$

Remarks:

- The objective (3.40) is to minimize the maximum execution time of all sub-tasks, which also minimizes the completion time of the input job.
- Constraints (3.41) and (3.42) are energy consumption constraints and resource availability constraints.
- Constraints (3.43) ensure that the total load of all sub-tasks of T_i equals to the load of task T_i, and constraints (3.44) are domain constraints.

We observe that in the SJS-Relax-Divisible problem, all constraints are affine and the objective is the pointwise maximum of NM ratios of affine functions. Therefore, this problem is a Generalized Linear Fractional Programming (GLFP) problem, which can be solved as a sequence of LP feasibility problems [31].

However, solving a GLFP problem can be time-consuming as it needs to

solve a set of LP feasibility problems. To avoid this, we further transform the SJS-Relax-Divisible problem into an equivalent LP problem.

An Equivalent LP Problem: SJS-Relax-LP. The transformation starts from defining variable T as the JCT, i.e.,

$$T = \max_{\substack{i=1,\ldots,N \\ j=1,\ldots,M}} \left\{ \frac{l_{ij}}{\lambda_{ij} c_{t_{ij}}} \right\}. \tag{3.45}$$

Substituting T into the objective function, we have

$$\begin{aligned} &Minimize \quad T \\ &T \geq \frac{l_{ij}}{\lambda_{ij} c_{t_{ij}}},\ i = 1, \ldots, N,\ j = 1, \ldots, M. \end{aligned} \tag{3.46}$$

Further define variable p_{ij} as

$$p_{ij} = c_{t_{ij}} \cdot T, \tag{3.47}$$

and then reformulate the constraints (3.46) as

$$\lambda_{ij} \cdot p_{ij} \geq l_{ij},\ i = 1, \ldots, N,\ j = 1, \ldots, M. \tag{3.48}$$

On the other hand, by substituting p_{ij} into constraints (3.42), we have

$$\sum_{j=1}^{M} A_{jk} \sum_{i=1}^{N} p_{ij} \leq B_k \cdot T,\ k = 1, \ldots, R. \tag{3.49}$$

Integrating all transformations shown above together, we obtain an equivalent problem, named **SJS-Relax-LP**, as shown below.

SJS-Relax-LP

$$Minimize \;\; T \tag{3.50}$$

Subject to

$$\lambda_{ij} \cdot p_{ij} \geq l_{ij}, \;\; i = 1, \ldots, N, \;\; j = 1, \ldots, M, \tag{3.51}$$

$$\sum_{i=1}^{N} \sum_{j=1}^{M} \alpha_{VM_{t_{ij}}} \cdot \frac{l_{ij}}{\lambda_{ij}} \leq E_{MAX}, \tag{3.52}$$

$$\sum_{j=1}^{M} A_{jk} \sum_{i=1}^{N} p_{ij} \leq B_k \cdot T, \;\; k = 1, \ldots, R, \tag{3.53}$$

$$\sum_{j=1}^{M} l_{ij} = Load_i, \;\; i = 1, \ldots, N, \tag{3.54}$$

$$p_{ij} \geq 0, \;\; l_{ij} \geq 0, \;\; i = 1, \ldots, N, \;\; j = 1, \ldots, M. \tag{3.55}$$

Naturally, the SJS-Relax-LP problem is a Linear Programming problem as its objective function and all constraints are linear. Therefore, it can be efficiently solved by linear programming algorithms.

3.4.3 The Task Placing and Resource Allocation (TaPRA) Algorithm and Its Simplified Version: TaPRA-fast

Based on the relaxation SJS-Relax-LP, we propose an algorithm, called Task Placing and Resource Allocation (TaPRA), to solve the SJS problem.

Algorithm 2 The TaPRA Algorithm

1: **function** TaPRA(A_{jk},B_k,$Load_i$,E_{MAX})
   Phase I:
2: Solve SJS-Divisible and get $\{l_{ij}, p_{ij}\}$
   Phase II:
3: **for** each $T_i \in J$ **do**
4: $u \leftarrow \text{argmax}_{k=1,\ldots,M}\{l_{ik}\}$;
5: $x_{ij} \leftarrow 1$ if $j == u$; otherwise, $x_{ij} \leftarrow 0$;
6: **end for**
7: Get $\{c_{t_i}, et_{t_i}\}$ using Equation (3.17); Update E_{total};
8: **if** $E_{total} > E_{MAX}$ **then** call REC;
   Phase III:
9: **while** true **do**
10: $\mathcal{T}_{max} \leftarrow \{T_i \mid et_{t_i} == JCT\}$;
11: **for** each $T_i \in \mathcal{T}_{max}$ **do**
12: $\mathcal{TM}_{valid} \leftarrow \{\text{all valid } TM_{ij}\}$;
13: **if** $\mathcal{TM}_{valid} == \emptyset$ **then continue;**
14: $TM_{iu} \leftarrow \text{argmin}_{TM_{ij} \in \mathcal{TM}_{valid}}\{\text{new } et_{t_i}\}$;
15: Perform TM_{iu} and update $\{c_{t_i}, et_{t_i}\}$; **break;**
16: **end for**
17: **if** no task movement is performed **then break**;
18: **end while**
19: **return** x_{ij} and c_{t_i}
20: **end function**

The TaPRA algorithm has three phases. In the first phase, TaPRA obtains a solution of the SJS-Relax-LP problem; in the second phase, it determines an initial solution of the SJS problem based on the solution of the SJS-Relax-LP problem; in the last phase, it utilizes a local search procedure to further optimize the obtained initial solution. Algorithm 2 shows the pseudocode of TaPRA.

We now introduce the details of each phase.

Phase I: Solve the relaxed problem. The TaPRA algorithm starts from

solving the relaxed LP problem SJS-Relax-LP. Let the solution of SJS-Relax-LP be T, l_{ij}, and p_{ij}.

Phase II: Obtain an initial solution of the SJS problem. In this phase, TaPRA obtains an initial solution of the SJS problem from the solution of SJS-Relax-LP in two steps:

- First, it determines the value of variable x_{ij}, i.e., obtaining the task placement plan. Specifically, for each task T_i, TaPRA selects sub-task t_{iu} that gets the largest portion of T_i and assigns task T_i to server s_u. Such assignment can be presented as

$$x_{ij} = \begin{cases} 1, & \text{if } j = \underset{k=1,\ldots,M}{\operatorname{argmax}} \left\{ l_{ik} \right\} \\ 0, & \text{otherwise} \end{cases}, \ i = 1, \ldots, N. \tag{3.56}$$

  The intuition is that if a server gets a larger portion of task T_i, the result may be "closer" to the optimal solution by assigning T_i to that server.

- Second, the TaPRA algorithm determines the value of c_{t_i}, i.e., the resource allocation plan. Because the task placement has been determined in the first step, the SJS problem is naturally reduced to the OptRA problem. Therefore, the TaPRA algorithm simply utilizes Equation (3.17) to determine the value of c_{t_i}.

However, in some cases, the assignment (3.56) may lead to a violation of the energy consumption constraint (3.32), because the server that gets the largest percentage of a task may not be the one with the highest efficiency to execute that task, i.e., consumes the least energy to execute that task.

Algorithm 3 Reduce Energy Consumption (REC)

1: **function** REC(x_{ij},c_{t_i},A_{jk},B_k,$Load_i$,E_{MAX},E_{total})
2: **while** $E_{total} > E_{MAX}$ **do**
3: List $\mathcal{H} \leftarrow \{\}$;
4: **for** each $t_i \in J$ & each $S_j \in DC$ **do**
5: $TM_{ij} \leftarrow \{T_i, s_{src_i}, s_j, \Delta E_{ij}, \Delta JCT_{ij}\}$;
6: **if** $\Delta E_{ij} < 0$ **then** add TM_{ij} into $\mathcal{H}$;
7: **end for**
8: $TM_{uv} \leftarrow \text{argmin}_{TM_{ij} \in \mathcal{H}}\{\Delta JCT_{ij}\}$;
9: $x_{u,s_{src_u}} \leftarrow 0$ and $x_{uv} \leftarrow 1$;
10: Get $\{c_{t_i}, et_{t_i}\}$ using Equation (3.17); Update E_{total};
11: **end while**
12: **end function**

To resolve the problem, the TaPRA algorithm utilizes a procedure called Reduce Energy Consumption (REC) to reduce total energy consumption by performing task movement. A task movement is moving a task from one server to another server and is defined by $TM_{ij} = \{T_i, s_{src_i}, s_j, \Delta E_{ij}, \Delta JCT_{ij}\}$, where task T_i is moved from the original server s_{src_i} to server S_j; ΔE_{ij} and ΔJCT_{ij} are the difference of total energy consumption and JCT respectively between the two task placements. The REC procedure runs in iterations. In each iteration, it finds out all task movements that can reduce the total energy consumption and performs the one with the smallest ΔJCT_{ij}. If the new task placement satisfies the energy consumption constraint, the REC procedure stops; otherwise, it starts the next iteration. Algorithm 3 describes the REC procedure.

Phase III: Local search. In this phase, the TaPRA algorithm utilizes a local search procedure to further improve the initial solution obtained in phase II.

In this local search procedure, TaPRA runs in iterations. In each iteration, it starts with identifying all tasks with the largest execution time and putting them into a set named $\mathcal{T}_{max}$. Subsequently, TaPRA iteratively considers each task in the set $\mathcal{T}_{max}$. For each task $T_i \in \mathcal{T}_{max}$, the TaPRA algorithm calculates all valid task movements (i.e., the energy consumption constraint is not violated and the execution time of T_i is reduced after performing the movement.) and put them into a set $\mathcal{TM}_{valid}$; it then selects the task movement which reduces the execution time of task T_i most; in the following, TaPRA performs the selected task movement and update x_{ij} and c_{t_i}. If a task movement is performed, the TaPRA algorithm starts a new iteration of phase III.

If the TaPRA algorithm cannot improve the execution time of any tasks in $\mathcal{T}_{max}$ in some iteration, it then finishes and returns the current solution x_{ij} and c_{t_i}, as it cannot improve the JCT anymore.

3.4.3.1 TaPRA-fast: A Simplified Version of TaPRA

The TaPRA algorithm begins with solving the SJS-Relax-LP problem which is a LP problem. When the problem's scale is large enough, solving this problem can be time-consuming. On the other hand, we observe that the local search procedure in the TaPRA algorithm can be used to optimize any feasible schedules. With such observations, we propose TaPRA-fast, a simplified version of TaPRA with less time complexity.

TaPRA-fast has two phases: First, it obtains an initial solution; Second, it utilizes the local search procedure used in the TaPRA algorithm to optimize the initial solution. To obtain an initial solution, the TaPRA-fast algorithm

Algorithm 4 Online scheduler: OnTaPRA

```
1: procedure OnTaPRA(Current time curT)
2:   Add all jobs arriving at curT to Q_wait;
3:   T_finish ← all tasks finishing at curT;
4:   if T_finish ≠ ∅ then
5:     Release all computing resources allocated to T_finish;
6:     Call the DRC procedure;
7:   end if
8:   if curT mod ΔT_schedule == 0 then
9:     Release resource allocated in last DRC call;
10:    Use a scheduling policy to schedule jobs in Q_wait;
11:    Call the DRC procedure;
12:  end if
13: end procedure
```

places each task T_i on the server with the highest efficiency on executing this task, i.e., λ_{ij}. Subsequently, TaPRA-fast calculates c_{t_i} and JCT based on the determined task placement.

3.5 Online Scheduling

In the previous section, we have studied how to schedule a single job composed of a set of independent tasks with the goal of minimizing its completion time and have proposed algorithms to solve this problem.

However, in practice, jobs arrive the system in a time sequence and in a long term view, our goal is to minimize the average JCT of all arrived jobs. With this goal, it may be inefficient to address each of the arrived jobs individually. Motivated by this, we propose an online scheduler named OnTaPRA, which periodically schedules all arrived jobs together. Algorithm 4 shows the main logic of OnTaPRA.

Online Scheduler: OnTaPRA. The OnTaPRA scheduler puts each arrived job into a waiting queue Q_{wait} and keeps track of the waiting time of each job in Q_{wait}.

The OnTaPRA scheduler periodically schedule all jobs in the waiting queue together. To schedule the jobs in Q_{wait}, the OnTaPRA scheduler uses a scheduling policy named Shortest Job First (SJF) which is introduced later. While all scheduled jobs are placed on corresponding servers according their task placement plan, those jobs that fail to be scheduled stay in Q_{wait}.

After the jobs in the waiting queue are scheduled and placed on the servers, there may be residual computing capacity on those servers. These residual resources are actually wasted, as no job can utilize these computing resources until the next call of scheduling algorithm. To follow the work conservation rule, the OnTaPRA scheduler further uses a procedure named Distribute Residual Capacity (DRC) to temporarily distribute the residual computing capacity of each server to the tasks running on that server. In this way, all computing resources of a server will be in use as long as there are tasks running on it. On the other hand, in the next round of scheduling, that residual capacity temporarily distributed will be recollected and treated as the available capacity of the servers. The details of the DRC procedure are introduced later.

Moreover, whenever a task finishes, the computing resource allocated to that task will be released and temporarily distributed to other tasks on the same server by using the DRC procedure.

Scheduling Policy: Shortest Job First (SJF). The OnTaPRA scheduler uses the Shortest Job First (SJF) scheduling policy to address all jobs in

Algorithm 5 The SJF Scheduling Policy

1: **procedure** SJF(Current Waiting Queue Q_{wait})
2: $Q_{wait}^{new} \leftarrow \emptyset$;
3: **while** $Q_{wait} \neq \emptyset$ **do**
4: Perform the MAR test;
5: **if** test fails **then** Add Q_{wait} to Q_{wait}^{new}; **break**;
6: **for** each job $J_k \in Q_{wait}$ **do**
7: Calculate $\{x_{ij}^k, c_{T_i}^k, JCT_k\}$ using TaPRA;
8: **if** fails **then** Move J_k from Q_{wait} to Q_{wait}^{new};
9: **end for**
10: $J_u \leftarrow \text{argmin}_{J_k \in Q_{wait}}\{JCT_k\}$;
11: Execute J_u; Remove J_u from Q_{wait};
12: **end while**
13: $Q_{wait} \leftarrow Q_{wait}^{new}$;
14: **end procedure**

Q_{wait}. (Line 10 of Algorithm 4).

The SJF scheduling policy runs in iterations. In each iteration, the SJF policy begins with a test named **Minimum Available Resource (MAR)** which checks the total amount of available computing capacity in the data center (denoted by C_{avai}^{total}). If C_{avai}^{total} is lower than a certain percentage (denoted by $Max\_Percent$) of the total computing capacity of all servers (denoted by C_{DC}^{total}), i.e., if the following condition is satisfied:

$$C_{avai}^{total} \leq Max\_Percent \cdot C_{DC}^{total}, \tag{3.57}$$

the test fails and the SJF policy stops scheduling all jobs current in Q_{wait}. The intuition here is that when the amount of available resource is small, a job may get little resource allocated and thereby have a extremely long completion time. In such cases, it may be a better choice to keep the job

Algorithm 6 The DRC procedure

1: **procedure** DRC
2: **for** each server $S_j \in DC$ **do**
3: $C_{s_j}^{resi} \leftarrow$ residual capacity of S_j;
4: $\mathcal{T}_{s_j} \leftarrow$ tasks on S_j; $L_{s_j} \leftarrow$ total load of $\mathcal{T}_{s_j}$;
5: **for** each $T_i \in \mathcal{T}_{s_j}$ **do** $c_{t_i} \leftarrow c_{t_i} + C_{s_j}^{resi} \cdot \frac{L_{t_i}}{L_{s_j}}$;
6: **end for**
7: **end procedure**

waiting until more resources become available.

If the SJF policy passes the MAR test, it sorts the jobs in Q_{wait} in the decreasing order of their waiting time and calculates the JCT by using the TaPRA algorithm for each job J_i. If a job cannot be scheduled, that job is moved to a new waiting queue Q_{wait}^{new}. Subsequently, the job with the smallest JCT is executed according to its schedule and removed from Q_{wait}. In the following, SJF updates the available computing capacity of the servers and starts a new iteration. Once the waiting queue Q_{wait} becomes empty, the SJF scheduling policy set the new waiting queue Q_{wait}^{new} as current Q_{wait} and finishes. Algorithm 5 shows the SJF scheduling policy.

Work Conservation: Distribute Residual Capacity (DRC). The DRC procedure iterates all servers. For each server S_j, the residual capacity of S_j is proportionally distributed to all tasks running on S_j according the load of those tasks.

Because the residual resource is not utilized by any job, by distributing these resources, we essentially avoid resource wastage, improve the system utilization, and further accelerate the job completion. On the other hand, such distribution of resource is temporary: Once new jobs arrive, the dis-

tributed resource is recollected and can be allocated to newly arrived jobs. Note that current virtualization techniques already support dynamic scaling of CPU and RAM for VMs [63]. Algorithm 6 shows the DRC procedure.

3.6 Performance Evaluation

3.6.1 Performance of the TaPRA Algorithm

In this section, we evaluate the TaPRA algorithm through offline simulations. In the following, we present our simulation setup, evaluation metrics, comparing algorithms and simulation results.

3.6.1.1 Simulation Setup

In each single run of the simulation, we randomly generate an input job, a set of servers, and a set of scheduling constraints. The TaPRA algorithm is then called to schedule the job on the given servers.

Job. We randomly generate an input job with N independent tasks. The load $Load_i$ of each task T_i follows a uniform distribution in the range of $(0, 3600]$ seconds.

Data Center. For the data center, we use a FatTree [64] architecture. A y-array FatTree architecture contains y pods. Each pod contains $y/2$ racks and each rack has $y/2$ servers. As a result, there are in a total of $y^3/4$ servers.

In the simulation, we modeled the available resource of each server S_j, i.e., C_{s_j}, by the number of virtual CPUs (vCPUs) that can be hosted by that server. Specifically, C_{s_j} follows a uniform distribution between 0 and 10

vCPUs.

The efficiency matrices λ_{ij} follows a uniform distribution in the range of $[0.1, 1]$. Because it is unlikely that the execution efficiency of a task is lower than 0.1 in a modern data center environment, we exclude the range $(0, 0.1)$ from the possible value of execution efficiency, following similar setups found in existing work [12].

Resource Availability Constraints. A y-array FatTree architecture contains y pods. We generate one resource availability constraint for each pod.

Specifically, for pod k, we denote the set of servers in this pod by $\mathcal{S}_{Pod_k}$. For each server $S_j \in \mathcal{S}_{Pod_k}$, we set the coefficient A_{jPod_k} as 1; for all other servers, we set A_{jPod_k} as 0. Subsequently, we calculate the total available resources in pod k, denoted by C_{Pod_k}, using the following equation

$$C_{Pod_k} = \sum_{S_j \in \mathcal{S}_{Pod_k}} C_{s_j}. \tag{3.58}$$

We then generate the following constraint for pod k

$$\sum_{j=1}^{M} A_{jPod_k} \sum_{i=1}^{N} x_{ij} c_{t_i} \leq B_{Pod_k} = \beta_{pod} \cdot C_{Pod_k}, \tag{3.59}$$

where β_{pod} is a constant belonging to $(0, 1]$. Using this way, we generate y constraints corresponding to y pods.

Following a similar approach, we generate $y^2/2 + 1$ constraints for the $y^2/2$ racks plus the whole data center in the y-array FatTree architecture. Therefore, we have $1 + y + y^2/2$ resources availability constraints for each single run of the simulation. In the simulations, we set β_{DC}, β_{pod} and β_{rack}

as 0.2, 0.2 and 0.3 respectively.

Energy Consumption Constraint. To generate the maximum energy consumption E_{MAX}, we first calculate the total task load $Load_{total}$, which equals to $\sum_{i=1}^{N} Load_i$. Furthermore, we set the constant α_{VM_i} as 1 for each VM_i.

Because the execution efficiency is no larger than 1, according to Equation 3.10, the minimum possible energy consumption equals to $Load_{total}$. We then determine E_{MAX} by using a uniform distribution in $[Load_{total}, 1.1 \cdot Load_{total}]$.

Each data point in our simulation results is an average of 50 simulations performed on an Intel 2.5 GHz processor.

3.6.1.2 Evaluation Metrics

We use three metrics to evaluate our algorithms.

JCT. Because minimizing JCT for the input job is our objective, JCT is the most important metric.

Total allocated resources (vCPUs). This metric is the sum of the resources allocated to each task of the input job, which is also the total number of vCPUs allocated to the input job, according to our simulation setup. This metric shows how well an algorithm utilizes the available resources and is useful when analyzing the simulation results.

Running time. Running time of an algorithm is also important. It gives a sense of the scalability of that algorithm.

3.6.1.3 Comparison Algorithms

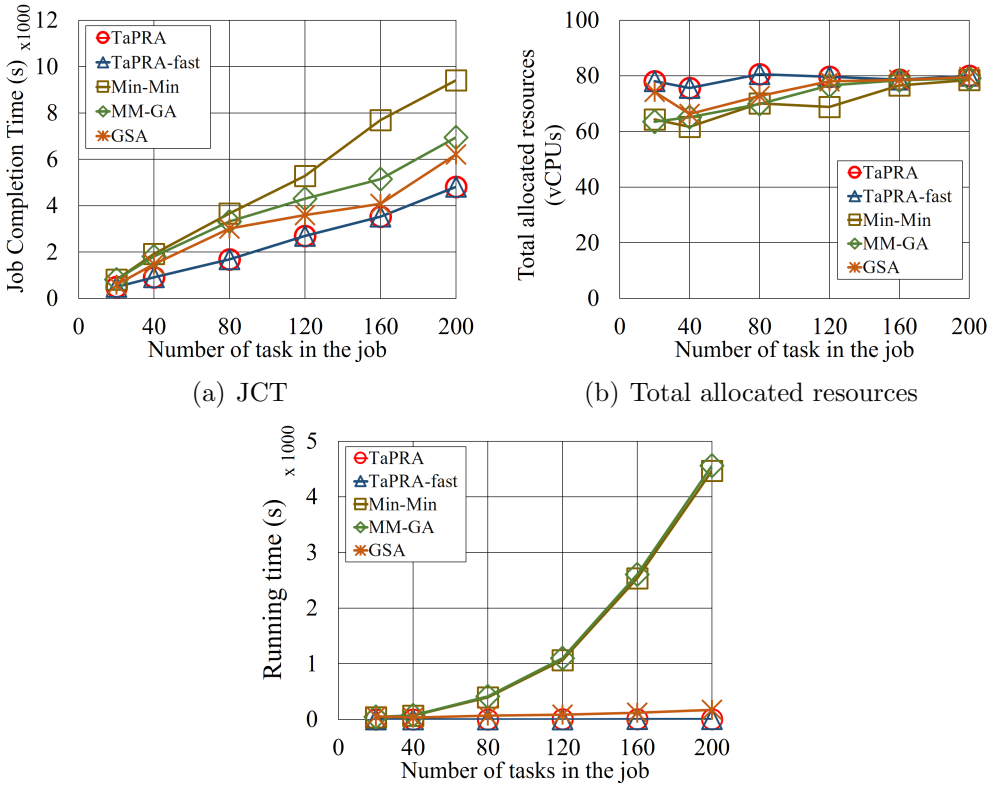
We compare our algorithms with three other algorithms.

Min-Min. The Min-Min algorithm is a classic scheduling algorithm. It runs in iterations. In each iteration, for each unplaced task, it calculates the expected JCT of placing that task on each server and adds the placement with the minimum expected JCT into a set $\mathcal{M}$. Subsequently, Min-Min selects and performs the placement with the smallest minimum expected JCT. It then starts a new iteration until all the tasks are placed.

MM-GA. Kumar *et al.* [41] proposed an improved genetic algorithm to schedule a set of independent tasks with the goal of minimizing the makespan. In that algorithm, the scheduling results of Max-Min and Min-Min are added into the initial population of the genetic algorithm. We use a modified version of this algorithm in which only the result of Min-Min is added into the initial population and we name this algorithm as MM-GA.

GSA. Gan *et al.* [42] proposed a genetic simulated annealing (GSA) algorithm which merged simulated annealing into a genetic algorithm. In each generation, the GSA algorithm generates a set of offspring using crossovers and mutations and then uses a simulated annealing procedure to further optimize those offspring.

Note that the above three algorithm are modified to consider the energy consumption constraint. Specifically, if a schedule generated by these algorithms violates the energy consumption constraint, the REC procedure is called to reduce the energy consumption for that schedule.



(a) JCT

(b) Total allocated resources

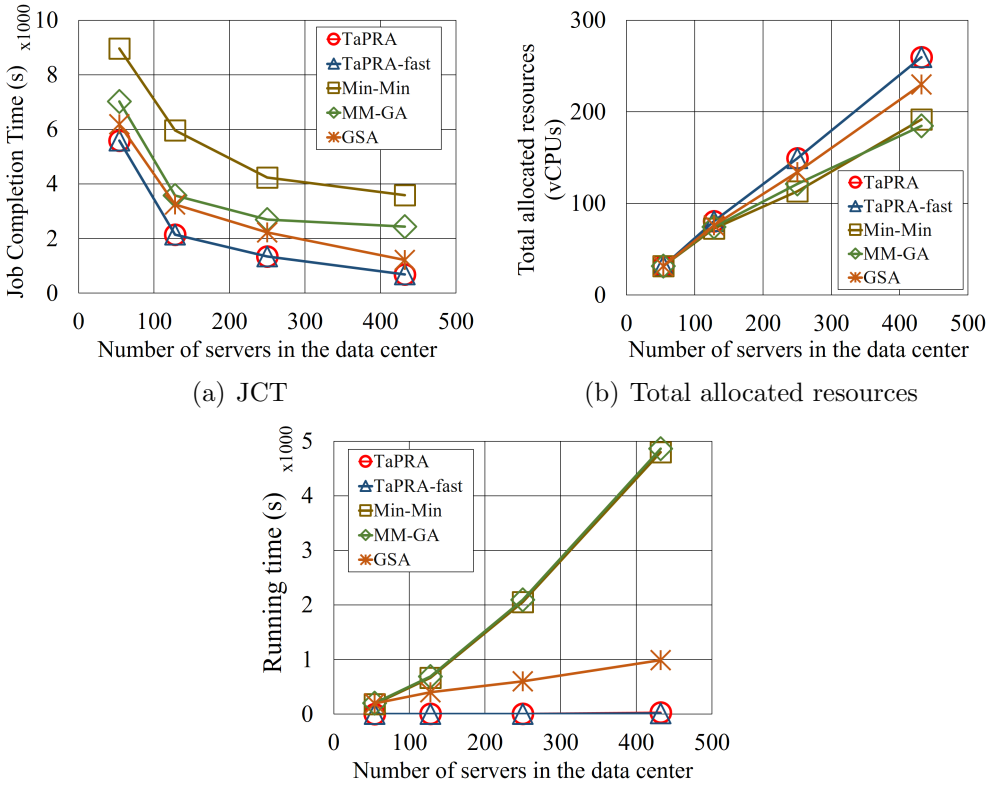
(c) Running time

Figure 3.4: Performance of TaPRA when scheduling a job with different numbers of tasks on a FatTree data center with 128 servers.

3.6.1.4 Evaluation Results of TaPRA and TaPRA-fast

Performance with Increasing Number of Tasks. In this simulation, we study how TaPRA and TaPRA-fast performs as the number of tasks in the job J increases from 40 to 200. We use a 8-array FatTree data center containing 128 servers.

Fig. 3.4(a) shows the JCT generated by each algorithm. While the JCT generally increases along with the expansion of the input job, TaPRA gen-



(a) JCT

(b) Total allocated resources

(c) Running time

Figure 3.5: Performance of TaPRA when scheduling a job with 100 tasks on a FatTree data center with different numbers of servers.

erates the smallest JCT. When the input job contains 200 tasks, the JCT of TaPRA is 100% smaller than that of Min-Min, 50% smaller than that of MM-GA, and 30% smaller than that of GSA. Meanwhile, TaPRA-fast generates almost the same JCT compared to TaPRA. Fig. 3.4(b) shows that the amount of resources allocated by each algorithm. We observe that when the number of tasks is small, TaPRA and TaPRA-fast allocate the largest amount of resources. When the number of tasks becomes large, all of the algorithms allocate similar amount of resources, however, TaPRA and TaPRA-fast gen-

erate much smaller JCT than Min-Min, MM-GA, and GSA. This observation shows that TaPRA and TaPRA-fast utilizes the available resources more efficiently. We attribute this to the analytical solution (3.17) proposed for the OptRA problem, which is used by TaPRA and TaPRA-fast to optimally allocate resource for any given task placement plan.

At last, Fig. 3.4(c) shows the algorithm running time. TaPRA and TaPRA-fast have much smaller running time than other algorithms. Specifically, when the job contains 100 tasks, the running time of TaPRA and TaPRA-fast is about 6 seconds, while that of GSA is 180 seconds and that of Min-Min and MM-GA is around 4500 seconds.

Performance with Increasing Number of Servers. In this simulation, we demonstrate how our algorithms perform as the number of servers in the FatTree data center increases from 54 to 432, i.e., the number of pods increases from 6 to 12. We fix the number of tasks in the job at 100.

Fig. 3.5(a) shows the JCT of TaPRA and TaPRA-fast We can see that the JCT of TaPRA is similar to that of TaPRA-fast, 80% smaller than that of GSA, 260% smaller than that of MM-GA, and 430% smaller than that of Min-Min. Fig. 3.5(b) shows the total amount of resources allocated by each algorithm. We observe that when the size of the data center is small, all algorithms allocate similar amount of resources, but when the data center becomes larger (containing more servers), TaPRA and TaPRA-fast allocate more resources than other algorithms. When the data center contains 432 servers, TaPRA and TaPRA-fast allocate around 15% more resources than GSA and about 40% more resoureces than Min-Min and MM-GA. One possible reason is that TaPRA and TaPRA-fast are able to generate better task

placement plan with more available resources to be allocated.

At last, Fig. 3.5(c) shows the algorithm running time. When the data center contains 12 pods (i.e., 432 servers), the running time of TaPRA is 33 seconds, while that of TaPRA-fast, GSA, Min-Min, and MM-GA is 15 seconds, 1000 seconds, 4700 seconds, and 4800 seconds respectively.

Summary. In offline simulations, we examine the performance of TaPRA and TaPRA-fast when scheduling a single input job. Generally, TaPRA and TaPRA-fast are able to place tasks in a way that more resources can be allocated and are able to utilize the allocated resources better. Benefiting from these properties, TaPRA and TaPRA-fast reduce the JCT by 40%-430% compared to the state-of-the-art algorithms. Moreover, the running time of these two algorithms are about 30 times faster than GSA and more than 200 times faster than Min-Min and GSA, which makes them more applicable in practice.

3.6.2 Performance of the OnTaPRA Scheduler

In this section, we examine the performance of the OnTaPRA scheduler through online simulations.

3.6.2.1 Simulation Setup

An online simulation starts from an initial state without any ongoing job. Subsequently, jobs start to arrive and the scheduler is called to schedule those jobs. Jobs stop arriving at a certain time. Once all jobs are scheduled, the online simulation finishes.

Data Center. A 6-array FatTree architecture with 54 servers is used in the simulation. The initial computing capacity of each server is 10 vCPUs.

Jobs. Jobs arrive at a rate following a Poisson distribution with $\mu = 0.15$/second and stop arriving at 3600 seconds.

Based on statistics of traces of workloads running on an 12000-server Google compute cell over a period approximately a month long, in May 2011 [65], we set the number of tasks in a job using a Weibull distribution [66] with scale parameter $A = 0.5$ and scale parameter $B = 0.8$. The maximum number of tasks in a job is set to 70. The task load follows a uniform distribution in $(0, 1000]$ seconds. The efficiency matrices λ_{ij} are generated using the same approach used in the offline simulations.

Constraints. The resources availability constraints and energy consumption constraints of a job J_i are generated whenever the scheduler is to schedule that job, using the same approach used in the offline simulations. Specially, we set β_{DC}, β_{pod}, and β_{rack} as 0.2, 0.2, and 0.3 respectively.

Each data point in the results is an average of 10 simulations performed on an Intel 2.5GHz processor.

3.6.2.2 Evaluation Metrics

Average Online JCT. The online JCT (JCT_{on}) of a job J_i is the length of the time period between the arriving time and the completion time of that job. It composed of the total waiting time of this job (denoted by JWT) and the offline JCT (JCT_{off}) generated by the scheduling algorithm, i.e.,

$$JCT_{on} = JWT + JCT_{off}. \tag{3.60}$$

Total Running Time. Total running time is the running time of a scheduler from the beginning of a simulation to the end.

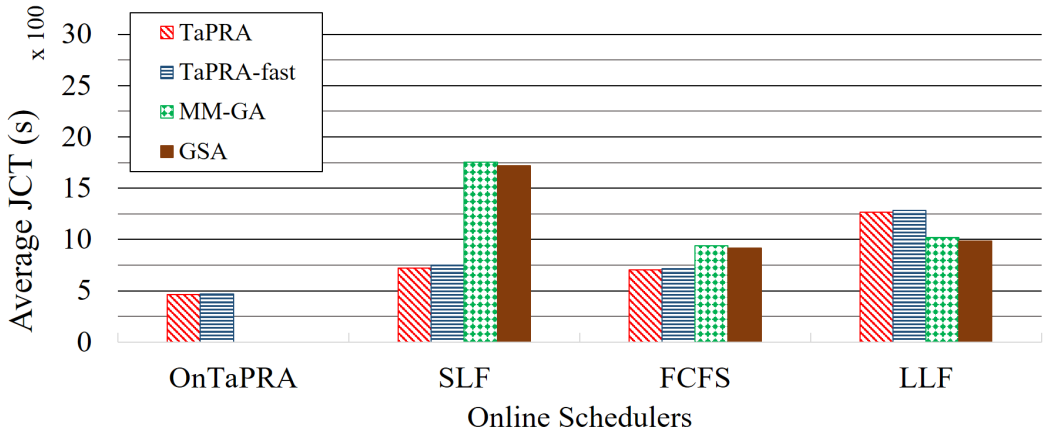
3.6.2.3 Comparison Schedulers and Algorithms

We compare OnTaPRA with three other schedulers.

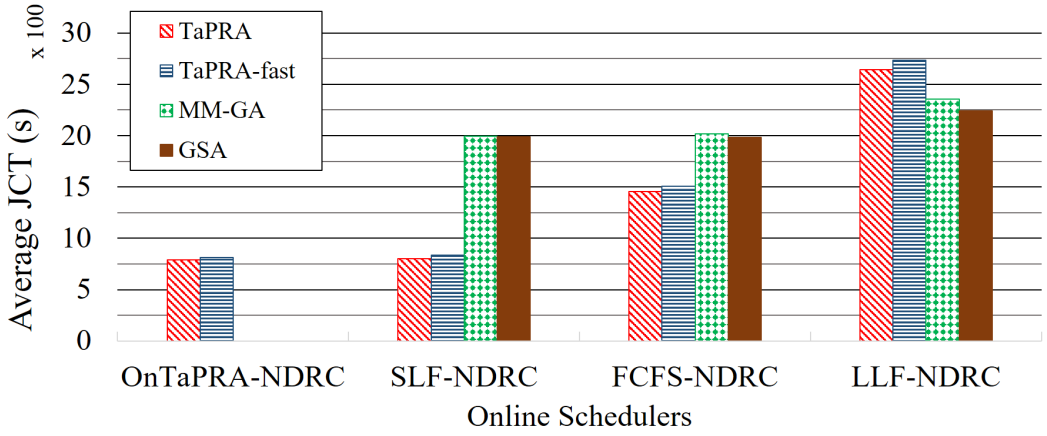
FCFS. The First Come First Serve (FCFS) scheduler is a classic scheduler which is still widely used in many scheduling systems because it is easily deployed and runs fast. In our simulations, whenever a job arrives the system, the FCFS scheduler address all waiting jobs in the decreasing order of their waiting time. If a job cannot be scheduled, it stays in the waiting queue; otherwise, it is executed according to the scheduling result. If the FCFS scheduler is DRC enabled, it then uses the DRC procedure to allocate residual capacity.

Smallest Load First (SLF). The SLF scheduler periodically address the jobs in Q_{wait} in the increasing order of their total load. We believe that generally a job's completion time is positively related to its total load, i.e., a small total load means a short JCT. Therefore, the SLF scheduler may be a good approximation of the SJF scheduling policy. By comparing to the SLF scheduler, we can better examine the performance of the OnTaPRA scheduler. The SLF scheduler can use the DRC procedure to allocate residual capacity.

Largest Load First (LLF). The LLF scheduler is a scheduler that periodically addresses the jobs in the waiting queue in the descending order of their total load, which is opposite to the SLF scheduling policy used in the OnTaPRA scheduler. The LLF schedule can also use the DRC procedure to



(a) DRC Enabled.



(b) DRC Disabled.

Figure 3.6: Average JCT generated by different schedulers in online simulations.

allocate residual capacity.

Comparison Algorithms. To make the comparison more comprehensive, each scheduler uses four different algorithms to schedule the arrived jobs, including TaPRA, TaPRA-fast, MM-GA and GSA. Min-Min is not included as MM-GA is guaranteed to be no worse than Min-Min.

3.6.2.4 Evaluation Results of OnTaPRA

Fig. 3.6 and Fig. 3.7 show the simulation results of each combination of schedulers and scheduling algorithms. Note that the MM-GA and GSA algorithms are not used by OnTaPRA, as their running time is relatively large; when using them, the running time of OnTaPRA is too large to be practical. To examine the impact of the DRC procedure, we also perform simulations for each scheduler without the DRC procedure. In the figures, the term "NDRC" appended after the scheduler name indicates that the scheduler is DRC disabled.

Fig. 3.6 shows the average JCT generated by each scheduler and Fig. 3.7 shows the running time of each scheduler. Based on the results, we have several observations:

Impact of Schedulers. Generally, the OnTaPRA scheduler generates the smallest average JCT. In the DRC enabled simulations, when using TaPRA or TaPRA-fast, the average JCT generated by OnTaPRA is about 60% smaller compared to the SLF and FCFS scheduler and about 170% smaller than the LLF scehduler. When compared to other schedulers using MM-GA or GSA, OnTaPRA reduces the average JCT by 100%-280%. In the DRC disabled simulations, when using TaPRA or TaPRA-fast, OnTaPRA generates 5%, 80%, and 234% smaller average JCT compared to SLF, FCFS, and LLF respectively.

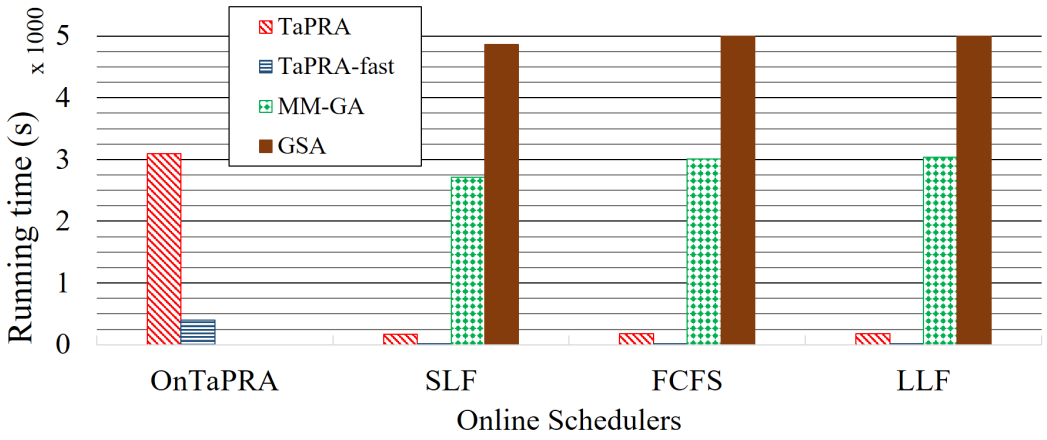
We can also see that when using TaPRA or TaPRA-fast, while SLF performs similar to FCFS and about 80% better than LLF in DRC enabled simulations, it performs 80% better than FCFS and 225% better than LLF. But when using MM-GA or GSA, SLF performs worse than FCFS and LLF

in DRC enabled simulations. These results show that the performance of SLF can be impacted by the DCR procedure and the scheduling algorithm. Whereas, the OnTaPRA scheduler always performs the best.

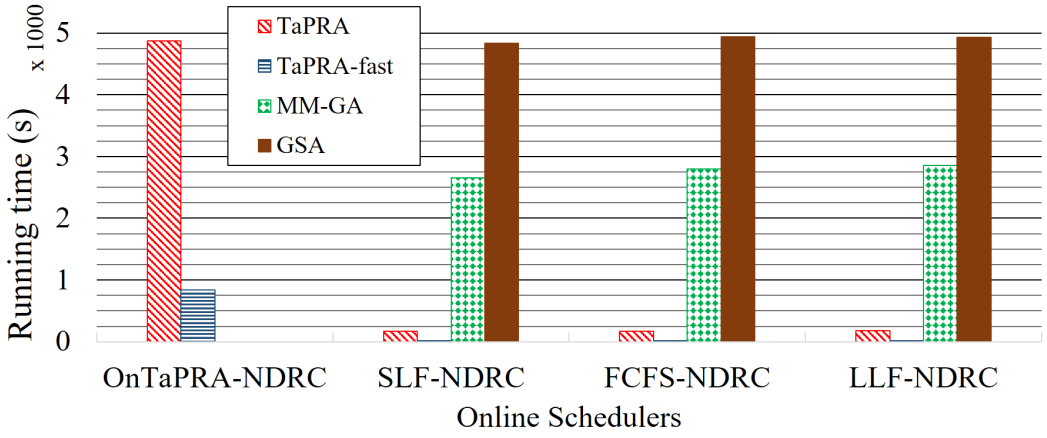
Impact of Scheduling Algorithm. Generally, TaPRA and TaPRA-fast show better performance than MM-GA and GSA, while TaPRA is about 5% better than TaPRA-fast. In DCR enabled simulations, TaPRA and TaPRA-fast reduces the average JCT by nearly 80% and 140% compared to MM-GA and GSA in the SLF and FCFS scheduler respectively, while in the LLF scheduler, TaPRA and TaPRA-fast performs actually worse than MM-GA and GSA. We attribute this to the better performance of TaPRA and TaPRA-fast on allocating resources because of which the LLF scheduler allocates most of the available resources to the larger jobs and therefore increases the completion time of the jobs with smaller load. We have similar observation in DRC disabled observations: TaPRA and TaPRA-fast performs 40%-150% better than MM-GA and GSA in the SLF and FCFS schedulers but worse than MM-GA and GSA in the LLF scheduler.

However, we can also see that the LLF scheduler performs worst among all schedulers; whereas, the OnTaPRA scheduler with TaPRA or TaPRA-fast performs the best.

Impact of the Distribute Residual Capacity (DRC) Procedure. The DRC procedure temporarily distributes the residual capacity of servers to the running tasks to accelerate the completion of running jobs. We can see that the DCR procedure has signification impact on the average JCT. Without the DCR procedure the average JCT generated by OnTaPRA, SLF, FCFS, and LLF is increased by 70%, 15%, 110%, and 120% respectively.



(a) DRC Enabled.



(b) DRC Disabled.

Figure 3.7: Running time of different schedulers in online simulations.

Scheduler Running Time. Fig. 3.7 shows the overall running time of each scheduler. Generally, the running time of using TaPRA and TaPRA-fast is much smaller than using MM-GA and GSA, which confirms our observation in offline simulations. Meanwhile, when using TaPRA or TaPRA-fast, the running time of OnTaPRA is larger than other schedulers, because of the

higher complexity of OnTaPRA. Furthermore, the running time of using TaPRA-fast is about 10 times faster than using TaPRA in all schedulers.

Summary. In online simulations, we demonstrate the performance of the OnTaPRA scheduler and the TaPRA/TaPRA-fast algorithms. The results show that: (a) the OnTaPRA scheduler with TaPRA/TaPRA-fast has the best performance: it reduces the average JCT to 60%-280% compared by existing schedulers; (b) the proposed DRC procedure has great impact on the average JCT: without this procedure, the average JCT can be increased by up to 120%; (c) while TaPRA-fast performs 5% worse than TaPRA, when using TaPRA-fast, the running time of OnTaPRA is 10 times smaller than using TaPRA, which makes OnTaPRA+TaPRA-fast an very applicable choice in practice.

3.7 Conclusion

In this paper, we focused on the problem of scheduling embarrassingly parallel jobs in cloud, in which there is a need to determine the task placement plan and the resource allocation plan for jobs composed of independent tasks with the goal of minimizing the Job Completion Time (JCT). We first studied how to optimally allocate resources with pre-determined task placement and proposed an analytical solution. In the following, we formulate the problem of scheduling a single job (SJS) as a NLMIP problem and present an relaxation with an equivalent Linear Programming problem. We further propose an algorithm named TaPRA and its simplified version: TaPRA-fast that solve the SJS problem. At last, to address multiple jobs in online scheduling, we

propose an online scheduler named OnTaPRA.

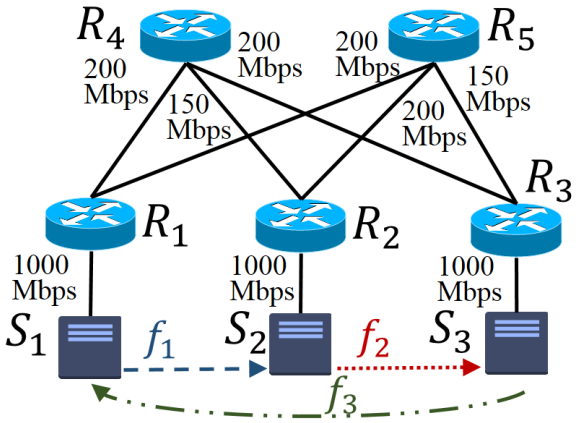
We evaluated the performance of the TaPRA and TaPRA-fast algorithms and the OnTaPRA scheduler by comparing them with the state-of-the-art algorithms and schedulers in offline and online simulations. The simulation results show that: (a) TaPRA and TaPRA-fast reduce the JCT by 40%-430% compared to the state-of-the-art algorithms and their running time is more than 30 times smaller. (b) The OnTaPRA scheduler when using TaPRA/TaPRA-fast reduces the average JCT by 60%-280% compared to exsiting schedulers. (c) TaPRA-fast can be 10 times faster than TaPRA with around 5% performance degradation compared to TaPRA, which makes the use of TaPRA-fast very applicable in practice.

Chapter 4

Coflow Scheduling in Data Centers: Routing and Bandwidth Allocation

4.1 Introduction

In recent years, we have witnessed a significant improvement on the IT infrastructures, like high performance computing systems, ultra high-speed networks and large-scale storage systems. Benefiting from these improvements, our ability of collecting, storing and processing data has also been dramatically enhanced. In a data center owned by big corporations like Google or Twitter, every day, hundreds terabytes of data can be transferred into its data storage system [67, 68], like HDFS [69] and GFS [70], and processed/analyzed by some computing frameworks such as MapReduce [71], Dyrad [72], Spark [73] and etc. By applying such data analysis on many



ID	Source	Desti--nation	Volume (Mb)
f_1	S_1	S_2	1200
f_2	S_2	S_3	400
f_3	S_3	S_1	200

Figure 4.1: Scheduling 3 flows in a network with 3 servers and 5 routers.

different areas like physics, biology, medicine, manufacture and finance, we have greatly changed the world we live in and made our life better. In such a context, one of the most important goals pursued by engineers and researchers is improving the execution performance of those data processing jobs.

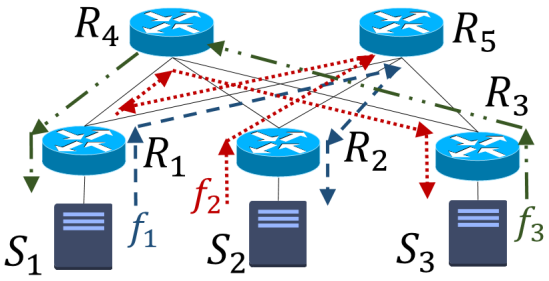
To achieve this goal, a critical problem to solve is how to optimize data transferring time. In many computing frameworks, jobs consist of a sequence of processing stages. Between two consecutive stages, there is usually a set of flows which move output data of the previous stage to the nodes executing the later stage. A job cannot start its next stage until all flows in this set finish. We usually refer such a set of flows as a *coflow* [74]. Since the transfer of a coflow can occupy more than 50% of the job completion time [75], optimizing the Coflow Completion Time (CCT) is important for improving the execution performance of jobs.

Recently many mechanisms [13–17] have been proposed to provide band-

width guarantee to network flows. With such ability, we can guarantee the flow completion time (FCT), i.e., the completion time of a single flow. However, scheduling a coflow in a fashion that minimizes its completion time is still a complex problem which involves both routing and bandwidth allocation at the level of the whole set of flows.

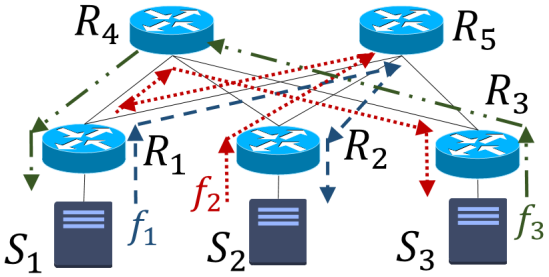
To illustrate this problem, consider the scenario shown in Fig. 4.1, in which a coflow with 3 individual flows (f_1, f_2, f_3) is waiting to be scheduled in a network with 3 servers and 5 routers. The goal is minimizing the CCT. To schedule this coflow, there is a need to determine a route for each flow and allocate a certain amount of bandwidth along each route. Fig. 4.2 shows three schedules generated by different scheduling strategies. Fig. 4.2(a) shows a schedule in which each flow is routed via the maximum capacity path and bandwidth is fairly allocated to flows using the same link, while Fig. 4.2(b) shows a schedule using the same routes but allocating bandwidth based on flow volume. We can see that by appropriately allocating bandwidth, the completion time of the largest flow f_1 is successfully reduced, which leads a decreases on the CCT. However, the schedule shown in Fig. 4.2(b) is not the optimal one: The determined routes share the same link, which causes bandwidth competition and limits the CCT. This is because routing is performed at the level of individual flows rather than the whole set of flows in this schedule. Whereas, Fig. 4.2(c) shows the optimal schedule in which we co-schedule all flows, reduce route overlap and allocate more bandwidth.

From this example, we can see that routing and bandwidth allocation together determine the CCT. We can obtain the minimum CCT only if we find out the optimal solution on both routing and bandwidth allocation.



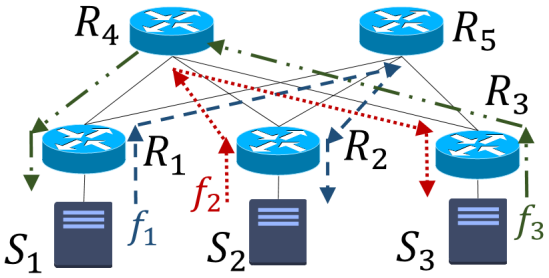
ID	Volume (Mb)	Band--width	FCT	CCT
f_1	1200	100 Mbps	12s	12s
f_2	400	100 Mbps	4s	
f_3	200	100 Mbps	2s	

(a) Schedule 1: Maximum capacity path + Fair sharing



ID	Volume (Mb)	Band--width	FCT	CCT
f_1	1200	150 Mbps	8s	8s
f_2	400	50 Mbps	8s	
f_3	200	20 Mbps	8s	

(b) Schedule 2: Maximum capacity path + Volume-proportional sharing



ID	Volume (Mb)	Band--width	FCT	CCT
f_1	1200	200 Mbps	6s	6s
f_2	400	100 Mbps	4s	
f_3	200	50 Mbps	4s	

(c) Schedule 3: Jointly consider routing and bandwidth allocation

Figure 4.2: Three flow schedules generated by different strategies.

However, in practice, we may need to schedule a large number of flows in a network with thousands or tens of thousands of nodes. In such cases, there exists a vast amount of possible routing plans and for each routing plan there are very many ways to allocate bandwidth. Searching the optimal solution

in such a huge solution space is not an easy problem to solve.

While several approaches have been proposed to schedule coflows [74–79], they either do not jointly consider routing and bandwidth allocation [74–78] or perform routing based on a limited set of candidate paths [79]. In this paper, we focus on the coflow scheduling problem in which we jointly consider routing and bandwidth allocation for a given coflow, formulate the problem as a non-linear programming problem, and propose algorithms to solve it.

In summary, our main contributions include

- We study the problem of optimal bandwidth allocation with pre-determined routes, in which the route of each flow in a given coflow has been determined and our goal is allocating bandwidth to each flow while minimizing the CCT. We formulate it as a convex optimization problem and provide an analytical solution. By solving this problem, we essentially reduce the dimension of the coflow scheduling problem: For any routing plan, we can always optimally allocate bandwidth. (Section 4.4)

- We formulate the coflow scheduling problem as a Mixed Integer Non-linear Programming (NLMIP) problem named CoS, which incorporates both routing constraints and bandwidth allocation constraints. We then present a relaxation of the CoS problem, named CoS-Relax, and transform it to a solvable convex optimization problem. (Section 4.5)

- We propose an algorithm, called Coflow Routing and Bandwidth Allocation (CoRBA), that solves the CoS problem based on the solution of CoS-Relax. With more practical consideration, we further propose CoRBA-fast, a simplified version of CoRBA with less complexity. (Section 4.6)

- We evaluate the performance of CoRBA and CoRBA-fast by comparing them with some existing algorithms through both offline and online simulations. The simulation results show that CoRBA can achieve 40%-500% smaller CCT than the existing algorithms. The results also show that CoRBA-fast can be hundreds times faster than CoRBA with up to 10% performance degradation, which makes it a good choice in practical use. (Section 4.7)

- We further discuss about the applicability of CoRBA-fast by comparing its running time with the transferring time of coflows in some typical MapReduce jobs. (Section 4.8)

4.2 Related Work

A significant amount of research has focused on the area of flow transmissions in data center networks. In this section, we discuss some of the works most relevant to our problems.

Coflow schedulers. The problem of coflow-aware flow scheduling has attracted significant attention [74–79]. Orchestra [75] is a global control architecture that manages the transfer of a set of correlated flows. In Orchestra, a set of algorithm has been proposed to improve the transfer time of such flows. Chowdhury *et al.* [74] explicitly propose the concept of coflow, a network abstraction that describes the traffic pattern of prevalent data flows. Barrat [76] is a decentralized flow scheduler which groups flows of a task and schedules them together with scheduling policies like FIFO-LM. Varys [77] is a coflow scheduler that addresses the inter-coflow scheduling problem, sup-

ports deadlines and guarantees coflow completion time. In Varys, a problem named concurrent open shop scheduling with coupled resources is introduced and heuristics are proposed to solve this problem. Aalo [78] is a recently proposed coflow scheduler that improves its predecessor Varys [77]. Aalo schedules coflows without any prior knowledge and it supports pipelining and dependencies in multi-stage DAGs. While the coflow schedulers introduced above focus on scheduling individual coflows or groups of coflows, however, they neglect routing, an important factor that impacts the coflow completion time.

RAPIER [79] is a recently proposed coflow-aware network scheduling framework that integrates both routing and bandwidth allocation. In RAPIER, scheduling a single coflow is formulated as a linear programming problem in which the route of each flow in the coflow is selected from a set of candidate paths given as input. In contrast to RAPIER, in this chapter, we propose algorithms that consider the bandwidth availability of the whole network and route flows via the best paths instead of picking up a path from a set of candidates. We show the superior performance of our algorithms by comparing them with the optimization algorithm used in RAPIER.

Flow schedulers. Much research work has also been performed on reducing the average flow completion time [24–30]. Rojas *et al.* [24] give a comprehensive survey on existing schemes for scheduling flows in data center networks. PDQ [26] is a flow scheduling protocol which utilizes explicit rate control to allocate bandwidth to flows and enables flow preemption. pFabric [27] is a datacenter transport design that decouples flow scheduling from rate control, in which flows are prioritized and switches implement a very sim-

ple priority-based scheduling/dropping mechanism. Munir *et al.* [28] classify the strategies used by existing data center transports into three categories and propose PASE, a framework that uses these strategies together with an appropriate division of responsibilities. RepFlow [29] is a transport design that replicates each short flow. It transmits the replicated and original flows via different paths, which reduces the probability of experiencing long queueing delay and therefore decreases the flow completion time. PIAS [30] is an information-agnostic flow scheduling scheme minimizing the FCT by mimicking the Shortest Job First strategy without any prior knowledge of incoming flows. While these existing schemes can reduce the FCT by using different strategies, they are not coflow-aware and therefore have different optimization objective compared to our algorithms.

4.3 Problem Definition

In this paper, we consider scheduling a given coflow on a given data center network. We now introduce the input, output, and objective of this problem.

Input. The input contains a data center network and a coflow. We model the data center network as a graph $\mathcal{G} =< \mathcal{V}, \mathcal{E}, \mathcal{B} >$, where $\mathcal{V}$ is the set of nodes, in which each server and router corresponds to one node; $\mathcal{E}$ is the set of links, in which a link E_{uv} presents the link between node u and node v; and $\mathcal{B}$ is the set of available bandwidth of links in $\mathcal{E}$, in which B_{uv} presents the available bandwidth of the link E_{uv}. Note that each link in the set $\mathcal{E}$ is unique, i.e., a link between node u and v is modeled as either E_{uv} or E_{vu}, but not both.

The coflow is a set of flows. We denote it by $\mathcal{F} = \{F_1, F_2, \ldots, F_N\}$ and define each flow F_i as $\{S_i, D_i, V_i\}$, where S_i is the source node of this flow, D_i is the destination node of this flow and V_i is the data volume of this flow, i.e., the total amount of data to be transferred. Like prior works [25–27,76,77,79], we assume that the information of a coflow can be captured by upper layer applications [74] or using existing prediction techniques [80].

Output. The output contains a set of routes, one for each flow in $\mathcal{CF}$, and a certain amount of bandwidth allocated to each route. We define the set of routes as $\mathcal{P} = \{p_1, p_2, \ldots, p_N\}$ in which p_i is the route selected for flow F_i. We also define b_i as the amount of bandwidth allocated to the route p_i.

Objective. Our objective is minimizing the Coflow Completion Time that is the completion time of the last finished flow. Let ct_i denote the completion time of flow F_i, then our objective is

$$Minimize \quad \max_{i=1,\ldots,N} \left\{ ct_i \mid ct_i = \frac{V_i}{b_i} \right\}. \tag{4.1}$$

4.4 Optimal Bandwidth Allocation with Pre-determined Routes

We start from the problem of optimal bandwidth allocation with pre-determined routes. In this problem, the route of each flow has already been determined and we need to allocate bandwidth to these routes with the goal of minimizing the CCT. With the solution of this problem, we can optimally allocate bandwidth corresponding to any routing plan, which essentially reduces the dimension of the coflow scheduling problem.

To model this problem, we define X_{uv}^{i} as a binary constant whose value is 1 if link E_{uv} is on the path p_i; otherwise it is 0. We can express it as

$$X_{uv}^{i} = \begin{cases} 1, & \text{if link } E_{uv} \text{ is on the path } p_i, \\ 0, & \text{otherwise.} \end{cases} \tag{4.2}$$

We then formulate the Optimal Bandwidth Allocation (OptBA) problem as

OptBA

Objective:

$$Minimize \quad \max_{i=1,\ldots,N} \left\{ ct_i \mid ct_i = \frac{V_i}{b_i} \right\} \tag{4.3}$$

Subject to

$$\sum_{i=1}^{N} \left| X_{uv}^{i} \right| \cdot b_i \leq B_{uv}, \ \forall u, \forall v, that \ E_{uv} \in \mathcal{E}, \tag{4.4}$$

$$b_i \geq 0, \ i = 1, \ldots, N \tag{4.5}$$

Remarks:

- The objective of DSBA-OPT-BA (4.3) is to minimize the maximum completion time of data flows in set $\mathcal{F}$. Because all flows are simultaneously released, by minimizing this objective function, we essentially minimize the flow set completion time of $\mathcal{F}$.

- Constraints (4.4) are bandwidth availability constraints, which limit that for any link E_{uv}, the overall amount of bandwidth allocated to flows using this link cannot exceed the available bandwidth of this link.

- Constraints (4.5) are domain constraints ensuring the bandwidth allocated to each flow to be non-negative.

We now show that the OptBA problem is a convex optimization problem. We observe that the functions in constraints (4.4) and (4.5) are all affine on b_i and thus convex. On the other hand, the function ct_i is convex, because its second derivative is nondecreasing when b_i is larger than 0. Therefore, according to [31], the objective function, i.e., the pointwise maximum function of ct_i, is also a convex function. As a result, the OptBA problem is a convex optimization problem.

4.4.1 Analytical Solution

While existing convex optimization algorithms can be used to solve the OptBA problem, we develop an analytical solution which is more efficient. Specifically, we define b_{uv}^i as the amount of bandwidth allocated to flow F_i on link E_{uv} when we proportionally distribute the available bandwidth of link E_{uv} to all flows that are using this link, i.e.,

$$b_{uv}^i = \frac{V_i}{\sum_{k=1}^{N} X_{uv}^k V_k} B_{uv}. \tag{4.6}$$

Let $\vec{b^*}$ be an vector$\{b_1^*, b_2^*, \ldots, b_N^*\}$, in which b_i^* is the minimum value of all existing b_{uv}^i, i.e.,

$$\begin{aligned} \vec{b^*} &= \{b_1^*, b_2^*, \ldots, b_N^*\}, \\ \text{where } b_i^* &= \min_{E_{uv} \in \mathcal{E}_i} \{\frac{V_i}{\sum_{k=1}^{N} X_{uv}^k V_k} B_{uv}\}, \end{aligned} \tag{4.7}$$

in which $\mathcal{E}_i$ is the set of links on the route of flow F_i. We then have that the vector $\vec{b^*}$ is an optimal solution of the OptBA problem. To prove this, we first prove the following lemma.

Lemma 3. *Assume that for the vector $\vec{b^*}$ defined in Equation 4.7, flow F_p has the largest finish time, i.e., $ct_p^* = V_p/b_p^* = \max_{i=1,\ldots,N}\{ct_i^*\}$. Also assume that b_p^* obtains its value when link $E_{u^*v^*}$ is considered, i.e.,*

$$b_p^* = \min_{E_{uv} \in \mathcal{E}_p} \left\{b_{uv}^p\right\} = b_{u^*v^*}^p = \frac{V_p}{\sum_{k=1}^{N} X_{u^*v^*}^k V_k} B_{u^*v^*}. \tag{4.8}$$

Then for every other flow F_i using link $E_{u^*v^*}$, its allocated bandwidth b_i^* also obtains its value when link $E_{u^*v^*}$ is considered, i.e,

$$b_i^* = \min_{E_{uv} \in \mathcal{E}_i} \left\{b_{uv}^i\right\} = b_{u^*v^*}^i = \frac{V_i}{\sum_{k=1}^{N} X_{u^*v^*}^k V_k} B_{u^*v^*}. \tag{4.9}$$

Proof. To begin with, we assume that there exists a flow F_q using link $E_{u^*v^*}$ but getting its allocated bandwidth when another link $E_{u'v'}$ is considered, i.e.,

$$b_q^* = \min_{E_{uv} \in \mathcal{E}_q} \left\{b_{uv}^q\right\} = b_{u'v'}^q = \frac{V_q}{\sum_{k=1}^{N} X_{u'v'}^k V_k} B_{u'v'}. \tag{4.10}$$

Next, we define ct'_q as the completion time of flow F_q when its allocated bandwidth equals to the bandwidth allocated on link $E_{u^*v^*}$, i.e.,

$$ct'_q = \frac{V_q}{b^q_{u^*v^*}}. \tag{4.11}$$

Based on Equations (4.10) and (4.11), we have

$$ct^*_q = \frac{V_q}{b^q_{u'v'}} > \frac{V_q}{b^q_{u^*v^*}} = ct'_q. \tag{4.12}$$

On the other hand, based on Equation (4.8), we have

$$ct^*_p = \frac{V_p}{b^p_{u^*v^*}} = \frac{V_q}{b^q_{u^*v^*}} = ct'_q. \tag{4.13}$$

Now using the Inequity (4.12) and Equation (4.13), we get

$$ct^*_q > ct'_q = ct^*_p, \tag{4.14}$$

which conflicts with the assumption the flow F_p has the largest completion time. Therefore, there does not exist a flow F_q and a link $E_{u'v'}$ that satisfy the Equation (4.10). As a result, we have proved the lemma. □

Based on Lemma 3, we have the following theorem.

Theorem 3. *The vector $\vec{b}^*$ defined in Equation 4.7 is an optimal solution of the DSBA-OPT-BA problem.*

Proof. Assume that flow F_p has the largest completion time and b^*_p obtains its value then link $E_{u^*v^*}$ is considered. Then according to Lemma 3, for every

flow using link $E_{u^*v^*}$, there exists

$$b_i^* = b_{u^*v^*}^i = \frac{V_i}{\sum_{k=1}^{N} X_{u^*v^*}^k V_k} B_{u^*v^*}. \tag{4.15}$$

Therefore, for all these flows, we have

$$ct_i^* = \frac{V_i}{b_i^*} = \frac{V_p}{b_p^*} = ct_p^*. \tag{4.16}$$

Now assume that instead of $\vec{b^*}$, a vector $\vec{b'}$ is the optimal solution. Also assume that flow F_q has the largest completion time. We have

$$ct_q' = \max_{i=1,\ldots,N}\{ct_i'\} < \max_{i=1,\ldots,N}\{ct_i^*\} = ct_p^*. \tag{4.17}$$

Using Equations (4.16) and (4.17), we have

$$ct_i' < ct_q' < ct_p^* = ct_i^*,\ \forall i,\ that\ X_{u^*v^*}^i \neq 0. \tag{4.18}$$

Naturally, we have

$$b_i' > b_i^*,\ \forall i,\ that\ X_{u^*v^*}^i \neq 0. \tag{4.19}$$

On the other hand, from Equation (4.15), we can get

$$\sum_{i=1}^{N} X_{u^*v^*}^i b_i^* = B_{u^*v^*}. \tag{4.20}$$

Putting Equations (4.19) and (4.20) together, we have

$$\sum_{i=1}^{N} X^i_{u^*v^*} b'_i > \sum_{i=1}^{N} X^i_{u^*v^*} b^*_i = B_{u^*v^*}, \tag{4.21}$$

which conflicts with the assumption that $\vec{b'}$ is a feasible solution. As a result, there does not exist a feasible solution that is better than $\vec{b^*}$. Therefore, the vector $\vec{b^*}$ defined in Equation (4.7) is an optimal solution of the OptBA problem. □

4.5 The Coflow Scheduling (CoS) Problem

In this section, we study the coflow scheduling problem: We formulate it as a Mixed Integer Non-linear Programming (MINLP) problem, present its relaxed NLP problem, and further transform the relaxed NLP problem to a solvable convex optimization problem.

To formulate the coflow scheduling problem, we define variable x^i_{uv} as an integer variable with three possible values (-1, 0, and 1). If data is transferred from node u to node v for the flow F_i, then x^i_{uv} equals to 1; if data is transferred from node v to node u for the flow F_i, then x^i_{uv} equals to -1; if data is not transferred between node u and node v for flow F_i, then x^i_{uv} equals to 0. We can express it as To formulate the coflow scheduling problem, we define variable x^i_{uv} as an integer variable with three possible values (-1, 0, and 1). If data is transferred from node u to node v for the flow F_i, then x^i_{uv} equals to 1; if data is transferred from node v to node u for the flow F_i, then x^i_{uv} equals to -1; if data is not transferred between node u and node v

for flow F_i, then x_{uv}^i equals to 0. We can express it as

$$x_{uv}^i = \begin{cases} 1, & \text{if } F_i \text{ flows from node u to v,} \\ -1, & \text{if } F_i \text{ flows from node v to u,} \\ 0, & \text{otherwise.} \end{cases} \tag{4.22}$$

Note that for a link between node u and v, there exists either x_{uv}^i or x_{vu}^i, corresponding to the existence of either E_{uv} or E_{vu}. The coflow scheduling problem is then presented as

CoS

Variables

- b_i: bandwidth allocated to the path p_i selected for F_i.
- x_{uv}^i: an integer variable defined by Equation (4.22).
- ct_i: the completion time of flow F_i.

Constants

- N: the number of flows in the set $\mathcal{F}$.
- B_{uv}: the available bandwidth of the link E_{uv}.
- $\mathcal{N}(u)$: the set of neighbor nodes of node u.

Objective:

$$Minimize \quad \max_{i=1,\ldots,N} \left\{ ct_i \;\middle|\; ct_i = \frac{V_i}{b_i} \right\} \tag{4.23}$$

Constraints:

$$\sum_{w \in \mathcal{N}(s_i)} x^i_{ws_i} - \sum_{w \in \mathcal{N}(s_i)} x^i_{s_i w} = -1, \quad i = 1, \ldots, N, \tag{4.24}$$

$$\sum_{w \in \mathcal{N}(s_i)} x^i_{wd_i} - \sum_{w \in \mathcal{N}(s_i)} x^i_{d_i w} = 1, \quad i = 1, \ldots, N, \tag{4.25}$$

$$\sum_{w \in \mathcal{N}(u)} x^i_{wu} - \sum_{w \in \mathcal{N}(u)} x^i_{uw} = 0, \ \forall i, \forall u \notin \{s_i, d_i\}, \tag{4.26}$$

$$\sum_{i=1}^{N} \left| x^i_{uv} \cdot b_i \right| \leq B_{uv}, \ \forall u, \forall v, that \ E_{uv} \in \mathcal{E}, \tag{4.27}$$

$$b_i \geq 0, \ i = 1, \ldots, N, \tag{4.28}$$

$$x^i_{uv} \in \{-1, 0, 1\}, \ \forall i, \ \forall u, \ \forall v. \tag{4.29}$$

Remarks:

- Constraints (4.24) ensure that data is sent out from the source of any flow through only one link. Because a positive value of $x^i_{ws_i}$ and a negative value of $x^i_{s_i w}$ mean that data is going into the source node via link $E_{s_i w}$ or E_{ws_i}, making the term $\sum_{w \in \mathcal{N}(s_i)} x^i_{ws_i} - \sum_{w \in \mathcal{N}(s_i)} x^i_{s_i w}$ to be -1 essentially restricts that only one link is selected to sent data out from the source node and there is no data transferred into the source node. Similarly, constraints (4.25) ensure that data is transferred into the destination node of any flow through only one link.

- Constraints (4.26) ensure flow conservation, i.e., for any flow F_i and any intermediate node u, the number of links through which data is

transferred into node u should be equal to the number of links through which data is sent out from node u. Note that constraints (4.24), (4.25) and (4.26) together enforce that only one path is selected to transfer data for each flow.

- Constraints (4.27) enforce that the overall bandwidth allocated to flows in coflow $\mathcal{CF}$ on any link E_{uv} does not exceed the total available bandwidth of that link.

- Constraints (4.28) and (4.29) are domain constraints.

We observe that x_{uv}^i is an integer variable, and the objective function and the constraint (4.27) are non-linear. Hence, the CoS problem is a NLMIP problem, which is hard to solve.

4.5.1 A Relaxation of the CoS Problem and An Equivalent Convex Optimization Problem

To solve the CoS problem, we first consider its relaxation in which the integer variable x_{uv}^i is relaxed to a real variable, i.e., changing the constraint (4.29) to

$$-1 \leq x_{uv}^i \leq 1,\ \forall i,\ \forall u,\ \forall v. \tag{4.30}$$

We name the relaxed problem as **CoS-Relax**.

Subsequently, we transform the CoS-Relax problem into an equivalent convex optimization problem. We start from defining variable T as the CCT, i.e.,

$$T = \max_{i=1,\ldots,N} \left\{ \frac{V_i}{b_i} \right\}.$$

We can then transform the objective function in the CoS-Relax problem into the following format

$$Minimize \;\; T$$
$$\frac{V_i}{b_i} \leq T, \;\; i = 1, \ldots, N. \tag{4.31}$$

We further define variable q_i as

$$q_i = T \cdot b_i, \tag{4.32}$$

and then reformulate the constraint (4.31) as

$$q_i \geq V_i, \;\; i = 1, \ldots, N. \tag{4.33}$$

Meanwhile, by substituting q_i into constraint (4.27), we have

$$\sum_{i=1}^{N} \left| x_{uv}^i \cdot q_i \right| \leq T \cdot B_{uv}, \;\; \forall u, \forall v, that \; E_{uv} \in \mathcal{E}. \tag{4.34}$$

Next, we define variable p_{uv}^i as

$$p_{uv}^i = x_{uv}^i \cdot q_i. \tag{4.35}$$

By substituting p_{uv}^i into constraint (4.34), we have

$$\sum_{i=1}^{N} \left| p_{uv}^i \right| \leq T \cdot B_{uv}, \;\; \forall u, \forall v, that \; E_{uv} \in \mathcal{E}. \tag{4.36}$$

Putting all transformations shown above together and substituting x_{uv}^i by p_{uv}^i/b_i, we get an equivalent problem of CoS-Relax, named CoS-Relax-Cvx, as shown below.

CoS-Relax-Cvx

Objective:

$$Minimize \;\; T \tag{4.37}$$

Constraints:

$$q_i \geq V_i, \;\; i = 1, \ldots, N, \tag{4.38}$$

$$\sum_{w \in \mathcal{N}(s_i)} p_{ws_i}^i - \sum_{w \in \mathcal{N}(s_i)} p_{s_i w}^i = -q_i \;\; i = 1, \ldots, N, \tag{4.39}$$

$$\sum_{w \in \mathcal{N}(s_i)} p_{wd_i}^i - \sum_{w \in \mathcal{N}(s_i)} p_{d_i w}^i = q_i \;\; i = 1, \ldots, N, \tag{4.40}$$

$$\sum_{w \in \mathcal{N}(u)} p_{wu}^i - \sum_{w \in \mathcal{N}(u)} p_{uw}^i = 0, \;\; \forall i, \forall u \notin \{s_i, d_i\}, \tag{4.41}$$

$$\sum_{i=1}^{N} \left| p_{uv}^i \right| \leq T \cdot B_{uv}, \;\; \forall u, \forall v, that \;\; E_{uv} \in \mathcal{E}, \tag{4.42}$$

$$T \geq 0. \tag{4.43}$$

All constraints in the CoS-Relax-Cvx problem are affine, except the constraints (4.42) which are convex constraints, because the absolute value of p_{uv}^i is a convex function and the sum of convex functions is still convex. Therefore, the CoS-Relax-Cvx problem is a convex optimization problem

Algorithm 7 The CoRBA Algorithm

function CoRBA(Network $\mathcal{G} =< \mathcal{V}, \mathcal{E}, \mathcal{B} >$, Coflow $\mathcal{CF}$)
  Phase I:
  Solve CoS-Relax-Cvx and get $\{T',\ q'_i,\ {p'}^i_{uv}\}$;
  Calculate ${x'}^i_{uv}$ and b'_i using Equations (4.32) and (4.35);
  Phase II:
  for each $F_i \in \mathcal{CF}$ **do**
    $p_i^{MC} \leftarrow$ max capacity path using ${x'}^i_{uv}$ as link capacity;
    $x^i_{uv} \leftarrow 1$ if E_{uv} on p_i^{MC}; otherwise, $x^i_{uv} \leftarrow 0$;
  end for
  Calculate b_i using Equation (4.45);
  CCT $\leftarrow max_{i=1,\ldots,N}\{ct_i \mid ct_i = V_i/b_i\}$;
  Phase III:
  Update $\mathcal{B}$ according to x^i_{uv} and b_i;
  while true **do**
    $\mathcal{F}_{max} \leftarrow \{F_i \mid ct_i == CCT\}$;
    for each $F_i \in \mathcal{F}_{max}$ **do**
      $\mathcal{E}_{congest} \leftarrow \{E_{uv} \mid E_{uv}$ is on p_i & $B_{uv} == 0\}$;
      Add b_i back to $\mathcal{B}$ along route of F_i;
      Set each $E_{uv} \in \mathcal{E}_{congest}$ as unavailable;
      $p_i^{MC} \leftarrow$ new max capacity path;
      Reset x^i_{uv} according to new p_i^{MC};
      Re-calculate b_i, ct_i, and CCT;
      if ct_i is reduced **then break**;
      else Reverse all changes made for F_i;
    end for
    if no ct_i is reduced **then break**;
  end while
  return x^i_{uv}, b_i, and CCT;
end function

which can be solved by using convex optimization algorithms [31].

4.6 The Coflow Routing and Bandwidth Allocation (CoRBA) Algorithm and Its Simplified Version: CoRBA-fast

To solve the CoS problem, we propose an algorithm, called Coflow Routing and Bandwidth Allocation (CoRBA).

The CoRBA algorithm has three phases. In the first phase, CoRBA obtains a solution of the CoS-Relax problem by solving the CoS-Relax-Cvx problem, the equivalent convex optimization problem of CoS-Relax; in the second phase, it determines an initial solution of the CoS problem based on the solution of the CoS-Relax problem; in the last phase, it utilizes a local search procedure to further optimize the obtained initial solution. Algorithm 7 shows the pseudocode of CoRBA.

We now introduce the details of each phase.

Phase I: Solve the relaxed problem. To begin with, the CoRBA algorithm solves the CoS-Relax-Cvx problem by using convex algorithms [31], as this problem is a convex optimization problem. Let the solution of CoS-Relax-Cvx be T', q'_i, and ${p'}^i_{uv}$.

After obtaining the solution of CoS-Relax-Cvx, CoRBA calculates the solution of CoS-Relax, denoted by ${x'}^i_{uv}$ and b'_i, using Equations (4.32) and (4.35).

Phase II: Obtain an initial solution of the CoS problem. In this phase, the CoRBA algorithm obtains an initial solution of the CoS problem in two steps:

- First, it determines the value of variable x_{uv}^i, i.e., obtaining a route for each flow. Specifically, for each flow F_i, CoRBA uses ${x'}_{uv}^i$ as the capacity of link E_{uv} and find the max capacity path [81] (denoted by p_i^{MC}) between the source and destination of this flow. In the following, CoRBA sets x_{uv}^i as 1 for each link on the max capacity path and set its value as 0 for all other links, i.e.,

$$x_{uv}^i = \begin{cases} 1, & \text{if } E_{uv} \text{ is on the path } p_i^{MC}, \\ 0, & \text{otherwise.} \end{cases} \tag{4.44}$$

- Second, the CoRBA algorithm determines the value of b_i, i.e., the amount of bandwidth allocated to each flow. Because the route of each flow is already determined in the first step, the CoS problem is naturally reduced to the OptBA problem which is already solved in section 4.4. Therefore, according to Equation (4.7), CoRBA calculates the value of b_i as

$$b_i = \min_{E_{uv} \in \mathcal{E}_i} \{ \frac{V_i}{\sum_{k=1}^{N} |x_{uv}^k| V_k} B_{uv}. \}. \tag{4.45}$$

Phase III: Local search. In this phase, the CoRBA algorithm utilizes a local search procedure to further improve the initial solution obtained in phase II.

In this local search procedure, CoRBA runs in iterations. In each iteration, it starts with identifying all flows with the largest completion time and putting them into a set named $\mathcal{F}_{max}$. Subsequently, CoRBA iteratively

considers each flow in the set $\mathcal{F}_{max}$. For each flow F_i in $\mathcal{F}_{max}$, the CoRBA algorithm puts all links that are on the route of this flow and that do not have any available bandwidth (Due to the initial bandwidth allocation to flows) into set $\mathcal{E}_{congest}$; it then sets all links in $\mathcal{E}_{congest}$ as unavailable and find a new max capacity route for flow i; in the following, it temporarily changes the route of flow F_i to the new route and re-calculates b_i and CCT for current routing plan. If the completion time of flow F_i is reduced in the new solution, the CoRBA algorithm stops considering all other flows in the set $\mathcal{F}_{max}$ and starts a new iteration; otherwise, it reverts all temporary changes that it has made and moves to the next flow in the set $\mathcal{F}_{max}$.

If the CoRBA algorithm cannot improve the completion time of any flow in $\mathcal{F}_{max}$ in some iteration, it then finishes and outputs current solution as the final solution, as it cannot improve the CCT anymore.

4.6.1 CoRBA-fast: A Simplified Version of CoRBA

The CoRBA algorithm begins with solving the CoS-Relax-Cvx problem which a is convex optimization problem. When the problem's scale is large enough, solving this problem can be time-consuming. On the other hand, we observe that the local search procedure in the CoRBA algorithm can be used to optimize any feasible coflow schedules. With such observations, we propose CoRBA-fast, a simplified version of CoRBA with less time complexity.

CoRBA-fast has two phases: First, it obtains an initial solution; Second, it utilizes the local search procedure used in the CoRBA algorithm to optimize the initial solution.

To obtain an initial solution, for each flow F_i, the CoRBA-fast algorithm

set the route of this flow as the shortest maximum capacity path (i.e., the shortest path with the maximum capacity). Subsequently, CoRBA-fast calculates b_i and CCT based on the determined routes. Algorithm 8 shows the pseudocode of CoRBA-fast.

4.7 Performance Evaluation

4.7.1 Performance Evaluation through Offline Simulations

In this section, we evaluate the proposed algorithms, CoRBA and CoRBA-fast, through offline simulations. In the following, we present our simulation setup, evaluation metrics, comparing algorithms and simulation results.

4.7.1.1 Simulation Setup

In a single run of the offline simulation, the proposed algorithms are called to schedule a random coflow on a data center network.

Data center network. For the network, we use a modified FatTree [64] architecture. A k-array FatTree network has k pods, where each pod has $k/2$ Top-of-Rack (ToR) switches and $k/2$ aggregation switches. While in each pod the ToR and aggregation switches are interconnected as a complete bipartite graph, each ToR switch also connects a rack of $k/2$ hosts. In addition, there are $(k/2)^2$ core switches that connect the aggregation switches of all pods. In general, a k-array FatTree network is able to support $k^3/4$ hosts.

To better evaluate our algorithms, we decide to increase the oversub-

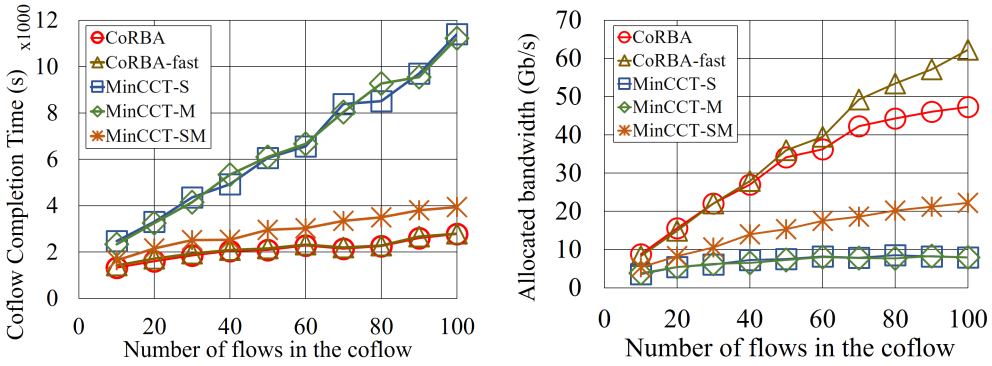
Algorithm 8 The CoRBA-fast Algorithm

1: **function** CoRBA-FAST(Network $\mathcal{G} =< \mathcal{V}, \mathcal{E}, \mathcal{B} >$, Coflow $\mathcal{CF}$)
   Phase I:
2: **for** each $F_i \in \mathcal{CF}$ **do**
3: $p_i^{MC} \leftarrow$ the shortest maximum capacity path of F_i;
4: $x_{uv}^i \leftarrow 1$ if E_{uv} on p_i^{MC}; otherwise, $x_{uv}^i \leftarrow 0$;
5: **end for**
6: Calculate b_i using Equation (4.45);
7: CCT $\leftarrow max_{i=1,\ldots,N}\{ct_i \mid ct_i = V_i/b_i\}$;
   Phase II:
8: Run phase III of CoRBA (Line 10–24 in Algorithm 7);
9: **return** x_{uv}^i, b_i, and CCT;
10: **end function**

scription ratio and therefore introduce more competition on bandwidth at the aggregation and core levels. To achieve this goal, we multiply the number of hosts in a rack by a factor α_{over}. By doing so, a k-array modified FatTree contains $\alpha_{over} \cdot k^3/4$ hosts now. In our simulations, we set α_{over} as 2 and set the each link's capacity as 10 Gbps.

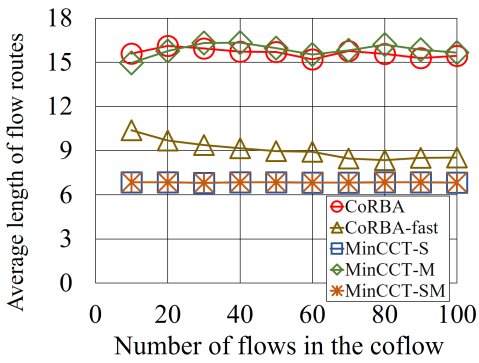
Noise flows. In order to simulate the complex traffic condition in real data center, we introduce noise flows into our simulations. Specifically, each single simulation run starts with generating a set of noise flows whose amount is 4 times of the number of hosts in the network. The source and destination of each noise flow is randomly selected and the duration follows an uniform distribution in $[1, 150]$. We randomly select a shortest path between as its route and allocate a certain amount of bandwidth which is the available bandwidth of the selected path multiplied by a random factor within $[0, 0.5]$.

Coflow. We randomly generate a coflow with N flows. For each flow, we randomly select two hosts as its source and destination. We further set the

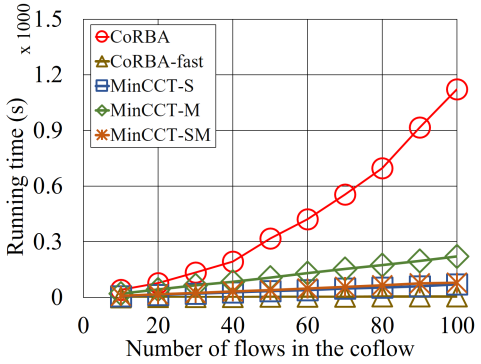


(a) Coflow Completion Time

(b) Allocated bandwidth



(c) Average length of flow route

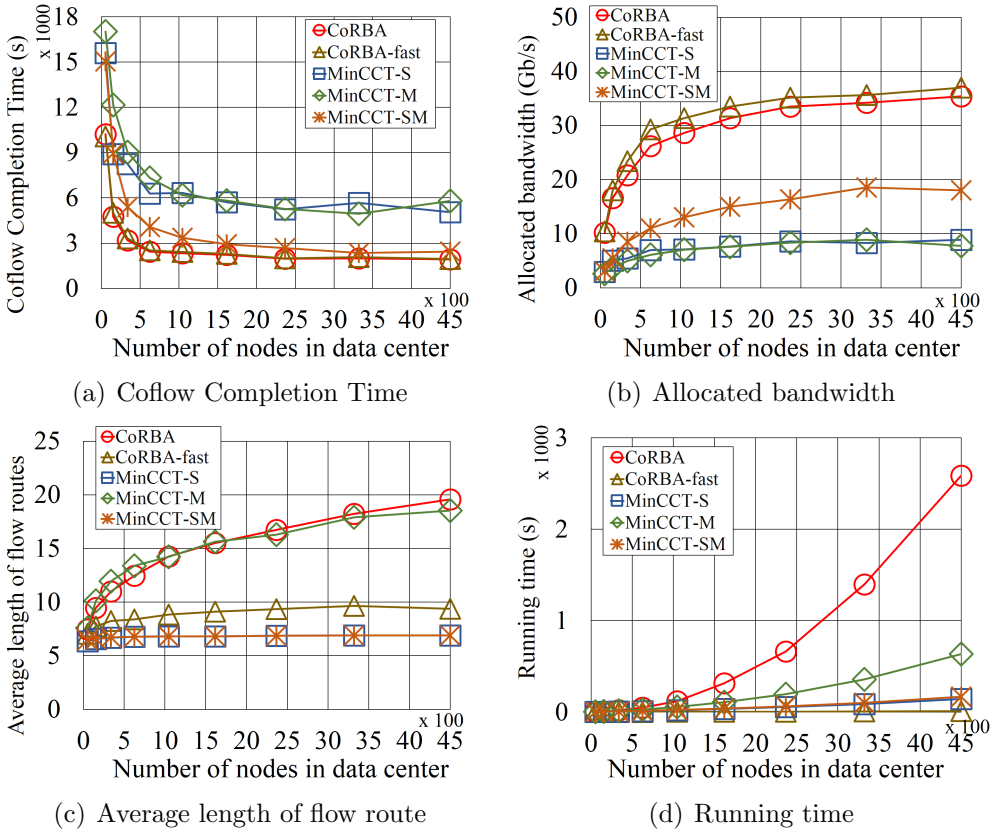


(d) Running time

Figure 4.3: Performance of CoRBA and CoRBA-fast when scheduling different numbers of flows in a FatTree network with 1617 nodes.

maximum possible volume of a flow (denoted by V_{max}) as 1000 Gb and determines the volume of a flow (i.e., V_i) by using an uniform distribution in $[\beta \cdot V_{max}, V_{max}]$. In our simulations, we set β as 0.7.

Each data point in our simulation results is an average of 20 simulations performed on an Intel 2.5 GHz processor.



(a) Coflow Completion Time

(b) Allocated bandwidth

(c) Average length of flow route

(d) Running time

Figure 4.4: Performance of CoRBA and CoRBA-fast when scheduling a coflow with 50 flows in FatTree networks with different number of nodes.

4.7.1.2 Evaluation Metrics

We use four evaluation metrics. **Coflow Completion Time.** This metric is the objective of the CoS problem. Hence, it is the most important metric. **Allocated bandwidth.** This metric is the overall bandwidth allocated to $\mathcal{CF}$, which equals to $\sum_{i=1}^{N} b_i$. It demonstrates how an algorithm performs on the aspect of bandwidth allocation. **Average length of flow route.** This metric is the average number of hops on the route of flows in $\mathcal{CF}$. It gives us

a sense about how an algorithm performs on the aspect of routing.

Running time. Running time of an algorithm is also important. It gives a sense of the scalability of that algorithm.

4.7.1.3 Comparison Algorithms

As a comparison algorithm of CoRBA, we use a modified version of the MinimizeCCT algorithm used in RAPIER [79]. Given a coflow, MinimizeCCT determines route for each flow and allocates bandwidth to these flows while minimizing the CCT. Although MinimizeCCT looks similar to the CoRBA algorithm, it does not perform true routing for the coflow. Instead, it selects each flow's route from a set of candidate paths given as input.

To compare the MinimizeCCT algorithm with our algorithm, we modify it to generate K paths as candidates for each flow before scheduling the coflow. We name this modified version as **MinCCT**. To make the comparison more comprehensive, we further propose three variants of MinCCT in which the K potential paths are (i) K shortest paths, (ii) K maximum capacity paths, and (iii) K shortest maximum capacity paths. We denote them by "MinCCT-S", "MinCCT-M", and "MinCCT-SM" respectively. In our simulations, we set the value of K as 5.

4.7.1.4 Evaluation Results of CoRBA

Performance with Increasing Size of the Coflow. In this simulation, we study how CoRBA and CoRBA-fast perform as the number of flows in the coflow increases from 10 to 100. We uses a 16-array modified FatTree with $\alpha_{over} = 2$, which contains 1617 nodes.

Fig. 4.3(a) shows the CCT generated by each algorithm. While the CCT generally increases along with the expansion of the coflow, CoRBA generates the smallest CCT. When the coflow contains 100 flows, the CCT of CoRBA is 300% smaller than that of MinCCT-M and MinCCT-S, and 40% smaller than that of MinCCT-SM. Meanwhile, CoRBA-fast generates almost the same CCT compared to CoRBA. We also observe that the growth rate of the CCT of CoRBA and CoRBA-fast is much lower than that of other algorithms. When the number of flows increases from 10 to 100, the CCT of CoRBA increases around 90%, but that of other algorithms increases 140%-380%.

Fig. 4.3(b) shows the total bandwidth allocated to the coflow. As can be seen, CoRBA allocates about 110%-500% more bandwidth than the MinCCT algorithms but about 25% less bandwidth than CoRBA-fast. Fig. 4.3(c) shows the average length of flow route. We observe that CoRBA and MinCCT-M generates the longest flow route length, followed by CoRBA-fast, MinCCT-SM and MinCCT-S.

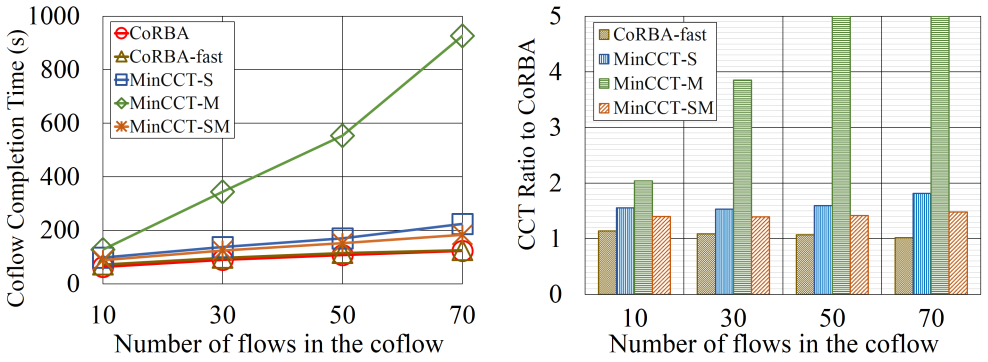
Putting Figs. 4.3(a), 4.3(b) and 4.3(c) together, we can see that when the size of the coflow increases, CoRBA and CoRBA-fast are able to allocate more bandwidth to the coflow and thereby keep the growth of CCT relatively flat, as observed in Fig. 4.3(a). We attribute this to their ability of routing based on the bandwidth availability of the whole network. When the coflow expands, CoRBA and CoRBA-fast are able to route flows via different paths and utilize more bandwidth. In contrast, the MinCCT algorithms are restricted by the limited number of candidate paths and cannot sufficiently utilize the available resources in the network, which leads the rapid increase of the CCT. In addition, we can also see that CoRBA-fast generates simi-

lar CCT with CoRBA while it allocates more bandwidth. This shows that CoRBA utilizes bandwidth more efficiently towards the goal of minimizing CCT.

At last, Fig. 4.3(d) shows the algorithm running time. When the coflow contains 100 flows, the running time of CoRBA is about 1100 seconds, while that of CoRBA-fast, MinCCT-S, MinCCT-M, and MinCCT-SM is 5 seconds, 70 seconds, 220 seconds, and 80 seconds respectively. One of the reasons is that CoRBA solves a programming problem containing a large number of variables, which is commonly a slow procedure. Another reason is the hardware limitation: All simulations are perform on a laptop with Intel i5-3250M CPU with 2.5 GHz speed. We believe that the algorithm can run much faster on servers with better CPUs. In addition, parallel convex optimization techniques have been well studied recently [82–84]. By utilizing such techniques, the CoRBA algorithm can be executed in a parallel environment and its running speed can be further improved.

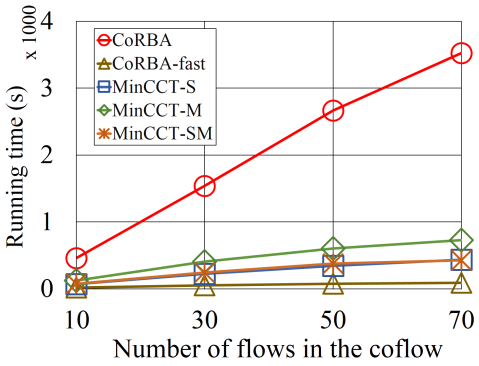
Performance with Increasing Size of the Network. In this simulation, we demonstrate how our algorithms perform as the number of nodes in the FatTree network increases from 52 to 4500, i.e., the number of pods increases from 4 to 20. We fix the number of flows in the coflow at 50.

Fig. 4.4(a) shows the CCT. We can see that the CCT of CoRBA is similar to that of CoRBA-fast, 30% smaller than that of MinCCT-SM, and 200% smaller than that of MinCCT-S and MinCCT-M. Fig. 4.4(b) shows the total allocated bandwidth. CoRBA and CoRBA-fast allocate similar amount of bandwidth, which is 2 times of that allocated by MinCCT-SM and 4 times of that allocated by MinCCT-M and MinCCT-SM.



(a) Coflow Completion Time

(b) Ratio CCT to CoRBA



(c) Running time

Figure 4.5: Performance of CoRBA and CoRBA-fast in online simulations with increasing number of flows in the coflow.

As can be seen, the bandwidth allocated by CoRBA is dramatically increased when the network just starts expanding and becomes more smooth thereafter. We attribute this to the limitation on how much bandwidth the algorithm can find to utilize. When the network is small, the number of good paths (with large available bandwidth) is also small. Consequently, CoRBA may schedule some flows to use paths with less bandwidth, which limits the CCT. At this stage, an expansion of network generates more good paths

and make CoRBA be able to allocate more bandwidth. However, along with the expansion, the number of good paths becomes more than enough and CoRBA is able to schedule most of the flows on these paths. At this stage, the quality of paths is barely improved. As a result, the increase of total allocated bandwidth becomes small. Such a trend on the allocated bandwidth then leads the trend of CCT, which is a steep decrease at the beginning but becomes relatively flat thereafter, as shown in Fig. 4.4(a).

We also observe that although MinCCT-M and MinCCT-SM use the maximum capacity paths as candidate routes, they actually allocate much less bandwidth than CoRBA. This is because when they generate candidate routes for a flow, they do not consider the candidate routes generated for other flows and the number of such routes is limited. As a result, it is possible that the routes of multiple flows overlap with each other, which leads bandwidth competition and therefore greatly limits the amount of bandwidth allocated to the coflow. Whereas CoRBA takes the whole network into consideration when performing routing and thereby is able to find routes with less or even no overlap and allocate more bandwidth.

Fig. 4.4(c) shows the average length of flow routes. MinCCT-M and CoRBA have the longest route length, followed by CoRBA-fast, MinCCT-S and MinCCT-SM. We attribute this to the fact that MinCCT-M and CoRBA route flows via the maximum capacity paths while MinCCT-S and MinCCT-SM route flows via the shortest paths.

At last, Fig. 4.4(d) shows the algorithm running time. When the network contains 16 pods (i.e., 4500 nodes), the running time of CoRBA is about 2500 seconds, while that of CoRBA-fast, MinCCT-S, MinCCT-M, and MinCCT-

SM is 10 seconds, 140 seconds, 600 seconds, and 160 seconds respectively.

Summary. In offline simulations, we examined the performance of CoRBA and CoRBA-fast when scheduling a single coflow. Benefiting from the ability of finding paths with less overlaps and allocating more bandwidth, CoRBA and CoRBA-fast outperforms MinCCT-SM by a factor of 1.3-1.5 and outperforms MinCCT-S and MinCCT-M by a factor of 3-5. Moreover, CoRBA-fast has similar performance with CoRBA but less efficiency on utilizing available bandwidth.

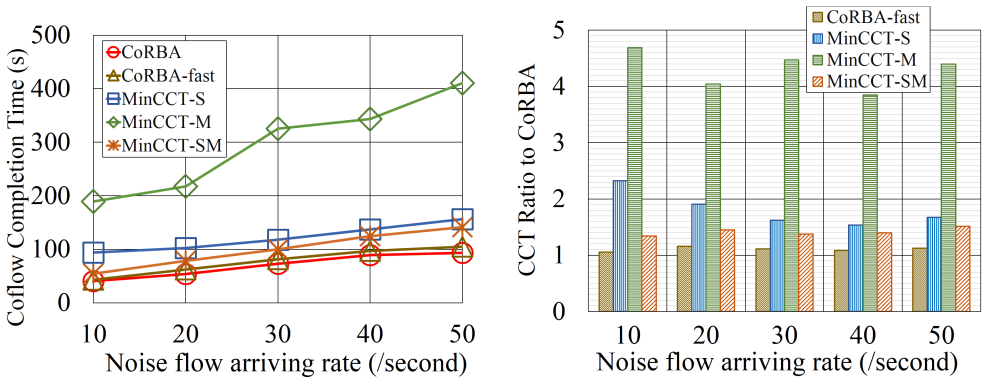
4.7.2 Performance Evaluation through Online Simulations

In this section, we examine long-term performance of the proposed algorithms through online simulations in which we simulate the execution of a flow scheduler using the state-of-the-art scheduling policy.

4.7.2.1 Simulation Setup

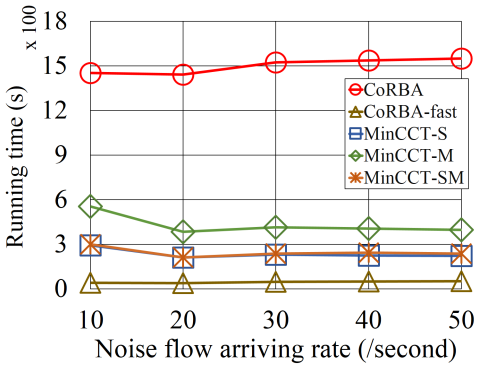
An online simulation begins with a 10-array modified FatTree network without any flows, in which there are 625 nodes and the link capacity is 10 Gbps. In the following, coflows start to arrive at a rate following a Poisson distribute with $\mu = 0.01$/second, and stop arriving at 1800 seconds. The whole simulation finishes, once all coflows are scheduled. Each data point in the results is an average of 10 simulations performed on an Intel 2.5GHz processor.

Scheduling Policy. We use the scheduling policy proposed in RAPIER [79] to schedule arrived coflows. Once a coflow arrives, the scheduling algorithm



(a) Coflow Completion Time

(b) Ratio CCT to CoRBA



(c) Running time

Figure 4.6: Performance of CoRBA and CoRBA-fast in online simulations with increasing arriving rate of noise flows.

is called to calculate the CCT of all existing coflows based on their current residual volume. In the following, all coflows are scheduled in the ascending order of their CCT. Once a coflow is scheduled, the CCT of all other coflows are re-calculated based on updated bandwidth availability. The selection procedure is repeated until all existing coflows are addressed. An existing coflow may be preempted by a newly arrived coflow. If scheduling a coflow is failed, this coflow is added into a waiting queue. Meanwhile, if a coflow has

been waiting for more than 100 seconds, it gets the privilege to be scheduled regardless of its calculated CCT. At last, once a coflow finishes, a special procedure is called to distribute the released bandwidth to existing coflows, aiming to facilitate the flow completion.

Noise flows. The noise flows arrives following a Poisson distribution and the duration of these noise flows follows an uniform distribution in $[1, 150]$. When a noise flow starts, we randomly selects a shortest path as its route and allocates a certain amount of bandwidth which is the link capacity times a random factor within $[0, 0.5]$.

4.7.2.2 Evaluation Metrics

We focus on two metrics: (i) the average CCT of coflows and (ii) the algorithm running time.

4.7.2.3 Comparison Algorithms

We compare our algorithms with MinCCT-S, MinCCT-M, and MinCCT-SM.

4.7.2.4 Evaluation Results of CoRBA

Performance with Increasing Size of the Coflow. We first study how our algorithms perform when the number of flows in the coflow increases. We set the arriving rate of noise flows to follow a Poisson distribution with $\mu = 40$/second.

Fig. 4.5(a) shows the average CCT generated by each algorithm and Fig. 4.5(b) shows the ratio of the average CCT of other algorithms to that of CoRBA. We observe that the average CCT generated by CoRBA is about

10%, 40%, 60%-80%, and 100%-650% smaller than that generated by CoRBA-fast, MinCCT-SM, MinCCT-S, and MinCCT-M respectively. Fig. 4.5(c) shows the running time of each algorithm. While CoRBA has the largest running time, CoRBA-fast has the smallest running time which is 4-8 times faster than the three MinCCT algorithms and about 40 times faster than CoRBA .

Performance with Increasing Size of the Coflow. We further study how our algorithms perform when the arriving rate of noise flows increases. We set each coflow containing 30 flows.

Fig. 4.6(a) shows the average CCT generated by each algorithm and Fig. 4.6(b) shows the ratio of the average CCT of other algorithms to that of CoRBA. We can see that the average CCT generated by CoRBA is about 10%, 40%, 60%–120%, and 400%-500% smaller than that generated by CoRBA-fast, MinCCT-SM, MinCCT-S, and MinCCT-M respectively. Fig. 4.6(c) shows the running time of each algorithm. Again, CoRBA has the largest running time and CoRBA-fast has the smallest running time which is 4-8 times faster than the three MinCCT algorithms and around 30 times faster than CoRBA.

Summary. We examined the performance of our algorithms in online simulations. The results show that the CoRBA and CoRBA-fast algorithms maintain their advantage over the three MinCCT algorithms: They are at least 40% (and up to 500%) better than the comparison algorithms. We attribute this to their better performance when scheduling each individual coflow.

On the other hand, while CoRBA-fast is much faster than CoRBA, CoRBA

performs about 10% better than CoRBA-fast in the online simulations. One possible reason is that CoRBA utilizes bandwidth more efficiently than CoRBA-fast when scheduling a single coflow, as shown in previous offline simulations. In long-term running, with such property, CoRBA is able to save more bandwidth and allocates them to other coflows. Therefore, it generates smaller average CCT.

4.8 Discussion

In simulations, it has been shown that CoRBA-fast can be tens of times faster than other algorithms, which makes it a good choice in practice. In this section, we further discuss about its applicability by comparing its running time with the transferring time of flows in some typical MapReduce jobs.

The shuffle stage in MapReduce jobs generates coflows. In shuffle-heavy jobs, like tera-sort and ranked-inverted-index, the shuffle volume can be more than 200 GB [85,86]. Considering the simulation case in which a coflow with 100 flows is scheduled in a network with 1617 nodes, the running time of CoRBA-fast is about 6 seconds and the bandwidth allocated to the coflow is around 60 Gb/s. In this case, if the coflow contains 200 GB data, its transferring time is about 27 seconds, which is nearly 5 times of the running time of CoRBA-fast. Whereas, the MinCCT algorithms allocate 10-20 Gb/s bandwidth and the coflow transferring time is 80-160 seconds. We can see that while CoRBA-fast significantly reduces the coflow transferring time, its running time is still generally small compared to the reduced transferring time. When examining other simulation cases, we get similar results. Therefore,

we believe that the use of CoRBA-fast is very applicable in practice.

4.9 Conclusion

In this paper, we focused on the coflow scheduling problem in which there is a need to perform routing and bandwidth allocation for a given coflow with the goal of minimizing the coflow completion time. We first studied how to optimally allocate bandwidth to a coflow with pre-determined routes. We formulated this problem as a convex optimization problem and provided an analytical solution with which we can always optimally allocate bandwidth corresponding to any routing plan. Subsequently, we formulated the coflow scheduling problem as a MINLP problem and presented a relaxation of this problem together with an equivalent convex optimization problem. To solve this problem, we proposed an algorithm called CoRBA and a simplified version of CoRBA called CoRBA-fast.

We evaluated the performance of CoRBA and CoRBA-fast by comparing them with some state-of-the-art algorithms in both offline and online simulations. In offline simulations, we examined how our algorithms perform when scheduling a single coflow. The simulation results shows that while CoRBA and CoRBA-fast have similar performance, they generate 30%-400% smaller CCT than their comparison algorithms. In online simulations, we simulated the execution of a flow scheduler and used the state-of-the-art scheduling policy. The results show that CoRBA and CoRBA-fast are at least 40% and up to 500% better than the comparison algorithms. Meanwhile, the results also show that CoRBA-fast can be tens of times faster than all other algorithms

with the cost of about 10% performance degradation compared to CoRBA, which makes CoRBA-fast very applicable in practice.

Chapter 5

Scheduling of Independent Flows in Data Centers: Routing and Bandwidth Allocation

5.1 Introduction

In the previous section, we studied the problem of scheduling coflows composed of a set of flows. While coflows represent one type of data transfers existing in the data centers, another type of data transfers are independent flows which have relatively large volume and can be initiated simultaneously. These flows are very common in data centers. Examples include data replication flows, database movements, distribution of experiment data. Due to their large volume, how to schedule these flows with the goal of minimizing their transfer time is a critical problem.

Similar to coflow scheduling, scheduling these independent flows also in-

cludes determining a routing plan and a bandwidth allocation plan. However, in contrast to coflow scheduling, the objective of this flow scheduling problem is to minimize their Total Transfer Time (TTT), equivalent to minimize the average Flow Completion Time (FCT). From the previous study of the coflow scheduling problem, we can see that how to efficiently schedule a set of flows with joint consideration of routing and bandwidth allocation is a complex problem to solve.

While several approaches have been proposed to reduce the average flow completion time [24–30], none of them considers routing and bandwidth allocation together. In this paper, we focus on the independent flow scheduling problem in which we jointly consider routing and bandwidth allocation for a given set of independent flows and propose offline algorithms and online scheduler to solve it.

In summary, our main contributions include

- We formally define the problem of scheduling independent flows, present the input, output, objective, and constraints. (Section 5.3)

- We study the problem of optimal bandwidth allocation with pre-determined routes, in which the routes of a given set of flows have been determined and our goal is allocating bandwidth to each flow while minimizing the TTT. We formulate it as a convex optimization problem. By solving this problem, we essentially reduce the dimension of the coflow scheduling problem: For any routing plan, we can always optimally allocate bandwidth. (Section 5.4.1)

- We formulate the problem of scheduling a single set of flows as a Mixed In-

teger Non-linear Programming (NLMIP) problem named SSFS, which incorporates both routing constraints and bandwidth allocation constraints. We then present a relaxation of the SSFS problem, named SSFS-Relax, and transform it to a solvable convex optimization problem. (Section 5.4.2)

- We propose an algorithm, called Flow Routing and Bandwidth Allocation (FRoBA), that solves the SSFS problem based on the solution of SSFS-Relax. With more practical consideration, we further propose FRoBA-fast, a simplified version of FRoBA with less complexity. (Section 5.4.3)

- We propose an online scheduler named OnFRoBA to address multiple flows in online scheduling. The OnFRoBA scheduler periodically schedules all flows in the waiting queue in two steps: (i) Selects a set of flow from the waiting queue; and (ii) calls scheduling algorithm to schedule the selected set of flows. We further proposed multiple flow approaches. For work conservation, the OnFRoBA scheduler distributes the residual bandwidth in the network to existing flows. (Section 5.5)

- We evaluate the performance of FRoBA and FRoBA-fast by comparing them with some existing algorithms through both offline and online simulations. The simulation results show that FRoBA can achieve 60%-250% smaller TTT than the existing algorithms. (Section 5.6)

5.2 Related Work

A significant amount of research has focused on the area of flow transmissions in data center networks. In this section, we discuss some of the research works

that we consider most relevant to our problems.

Many research work have been perform on reducing the average flow completion time (FCT) [24–30]. Rojas *et al.* [24] give a comprehensive survey about existing schemes for the transmission of flows in data center networks. D^3 [25] is a Deadline-Driven Delivery control protocol that makes data center networks deadline aware. It utilizes the explicit rate control to assign flows with rates based on their deadlines, instead of the fair share. PDQ [26] is a flow scheduling protocol which utilizes explicit rate control to allocated bandwidth to flows and enables flow preemption. In this way, it can transfer the most critical flows as quick as possible by preemptively allocating the required resources to them. pFabric [27] is a datacenter transport design that decouples flow scheduling from rate control. In this design, flows are prioritized and switches implements a very simple priority-based scheduling/dropping mechanism. The goal of rate control is then to avoid persistently high packet drop rates. Munir *et al.* [28] classify the strategies used by existing data center transports into three categories: self adjusting endpoints, arbitration and in-network prioritization. Munir *et al.* then propose PASE, a transport framework that uses these strategies together with an appropriate division of responsibilities. RepFlow [29] is a transport design that replicates each short flow. It transmits the replicated and original flows via different paths, which reduces the probability of experiencing long queueing delay and therefore decreases the flow completion time. PIAS [30] is a information-agnostic flow scheduling scheme that minimizes the FCT by mimicking the Shortest Job First strategy without any prior knowledge of incoming flows. While these existing schemes can reduce the FCT by using different strategies, they do

not consider optimizing the route of flows which has important impact on the flow completion time, as shown in previous example. In contrast to these schemes, we jointly consider routing and bandwidth allocation in this chapter and therefore achieve better performance.

5.3 Problem Definition

In this paper, we consider scheduling a given set of flows on a given data center network. We now introduce the input, output, and objective of this problem.

5.3.1 Input

The input contains a data center network and a set of input flows.

Data Center Network. We model the data center network as a graph $\mathcal{G} =< \mathcal{V}, \mathcal{E}, \mathcal{B} >$, where $\mathcal{V}$ is the set of nodes, in which each server and router corresponds to one node; $\mathcal{E}$ is the set of links, in which a link E_{uv} presents the link between node u and node v; and $\mathcal{B}$ is the set of available bandwidth of links in $\mathcal{E}$, in which B_{uv} presents the available bandwidth of the link E_{uv}. Note that each link in the set $\mathcal{E}$ is unique, i.e., a link between node u and v is modeled as either E_{uv} or E_{vu}, but not both.

The input flows. We denote the set of input flows by $\mathcal{F} = \{F_1, F_2, \ldots, F_N\}$ and define each flow F_i as $\{S_i, D_i, V_i\}$, where S_i is the source node of this flow, D_i is the destination node of this flow and V_i is the data volume of this flow, i.e., the total amount of data to be transferred. We assume that the information of the input flows can be captured by upper layer applications

or using existing prediction techniques [80].

5.3.2 Output

The output contains a routing plan and a bandwidth allocation plan.

Routing Plan. A routing plan is a set of routes, one for each flow in $\mathcal{F}$. We define the routing plan by $\mathcal{P} = \{P_1, P_2, \ldots, P_N\}$ in which P_i is the route selected for flow F_i. A route P_i contains a set of links which forms a path from source node S_i to destination node D_i. We use an integer variable x_{uv}^i to indicates the relationship between the route P_i and the link E_{uv}. Specifically, the variable x_{uv}^i has three possible values (-1, 0, and 1). If data is transferred from node u to node v for the flow F_i, then x_{uv}^i equals to 1; if data is transferred from node v to node u for the flow F_i, then x_{uv}^i equals to -1; if data is not transferred between node u and node v for flow F_i, then x_{uv}^i equals to 0. We can express it as

$$x_{uv}^i = \begin{cases} 1, & \text{if } F_i \text{ flows from node u to v,} \\ -1, & \text{if } F_i \text{ flows from node v to u,} \\ 0, & \text{otherwise.} \end{cases} \tag{5.1}$$

Note that for a link between node u and v, there exists either x_{uv}^i or x_{vu}^i, corresponding to the existence of either E_{uv} or E_{vu}.

Bandwidth Allocation Plan. A bandwidth allocation plan indicates the amount of bandwidth allocated to each flow. We denote the bandwidth allocation plan by $\vec{b} = \{b_{f_1}, \ldots, b_{f_N}\}$ in which b_{f_i} is the amount of bandwidth allocated to flow F_i.

5.3.3 Objective

Our objective is minimizing theTotal Transfer Time (TTT) that is the overall transfer time of all input flows. Let tt_i denote the transfer time of flow F_i, then our objective is

$$Minimize \;\; TTT = \sum_{i=1}^{N} tt_i = \sum_{i=1}^{N} \frac{V_i}{b_{f_i}}. \tag{5.2}$$

5.3.4 Constraints

Flow Conservation Constraints. For each flow F_i, only one path is selected to transfer data from the source S_i to the destination D_i. To achieve this, there should be three sets of constraints:

- Data should be sent out from the source of any flow through only one link. We can formulate these constraints as

$$\sum_{w \in \mathcal{N}(s_i)} x^i_{ws_i} - \sum_{w \in \mathcal{N}(s_i)} x^i_{s_iw} = -1, \quad i = 1, \ldots, N. \tag{5.3}$$

  Because a positive value of $x^i_{ws_i}$ and a negative value of $x^i_{s_iw}$ mean that data is going into the source node via link E_{s_iw} or E_{ws_i}, making the term $\sum_{w \in \mathcal{N}(s_i)} x^i_{ws_i} - \sum_{w \in \mathcal{N}(s_i)} x^i_{s_iw}$ to be -1 essentially restricts that only one link is selected to sent data out from the source node and there is no data transferred into the source node.

- Data should be transferred into the destination node of any flow through only one link. Similar to constraints (5.3), we can formulate these con-

straints as

$$\sum_{w \in \mathcal{N}(s_i)} x^i_{wd_i} - \sum_{w \in \mathcal{N}(s_i)} x^i_{d_i w} = 1, \quad i = 1, \ldots, N. \tag{5.4}$$

- For any flow F_i and any intermediate node u, the number of links through which data is transferred into node u should be equal to the number of links through which data is sent out from node u. We can formulate these constraints as

$$\sum_{w \in \mathcal{N}(u)} x^i_{wu} - \sum_{w \in \mathcal{N}(u)} x^i_{uw} = 0, \ \forall i, \forall u \notin \{s_i, d_i\}. \tag{5.5}$$

Constraints (5.3), (5.4) and (5.5) together enforce that only one path is selected to transfer data for each flow.

Bandwidth Availability Constraints. Bandwidth availability constraints ensure that the overall bandwidth allocated to the input flows on any link E_{uv} does not exceed the total available bandwidth of that link. We can formulate these constraints as

$$\sum_{i=1}^{N} \left| x^i_{uv} \cdot b_{f_i} \right| \leq B_{uv}, \ \forall u, \forall v, that \ E_{uv} \in \mathcal{E}. \tag{5.6}$$

5.4 Scheduling a Single Set of Flows

In this section, we focus on scheduling a single set of flows. We start from its sub-problem: Optimal Bandwidth Allocation with Pre-determined Routing Plan (OptBA-TTT).

5.4.1 Optimal Bandwidth Allocation with Pre-determined Routing Plan

In the OptBA-TTT problem, the routing plan has already been determined, i.e., the value of x_{uv}^i is known. For convenience and clarity, we use X_{uv}^i to indicate the determined routing plan. Our goal is to allocate available bandwidth to these flows while minimizing the TTT. With the solution of this problem, we can optimally allocate bandwidth corresponding to any routing plan, which essentially reduces the dimension of the coflow scheduling problem.

With the determined routing plan, the bandwidth availability constraints (5.6) becomes

$$\sum_{i=1}^{N} \left|X_{uv}^i\right| \cdot b_{f_i} \leq B_{uv},\ \forall u, \forall v, that\ E_{uv} \in \mathcal{E}. \tag{5.7}$$

We can then formulate the OptBA-TTT problem as

OptBA-TTT

$$Minimize\ \ TTT = \sum_{i=1}^{N} tt_i = \sum_{i=1}^{N} \frac{V_i}{b_{f_i}}. \tag{5.8}$$

Subject to

$$\sum_{i=1}^{N} \left|X_{uv}^i\right| \cdot b_{f_i} \leq B_{uv},\ \forall u, \forall v, that\ E_{uv} \in \mathcal{E}, \tag{5.9}$$

$$b_{f_i} \geq 0,\ i = 1, \ldots, N. \tag{5.10}$$

Remarks:

- Constraints (5.9) are bandwidth availability constraints; Constraints (5.10) are domain constriants.
- Flow conservation constraints (5.3), (5.4), and (5.5) are not included in OptBA-TTT, as they are only related with the variable x_{uv}^{i} whose value is already determined in the above problem.

We next show that the OptBA-TTT problem is a convex optimization problem. We observe that the function V_i/b_{f_i} is convex, because its second derivative is nondecreasing when b_{f_i} is larger than 0. Therefore, according to [31], the objective function, as the sum of a convex function, is also a convex function. In addition, the functions in constraints (5.9) and (5.10) are all affine on b_{f_i} and thus convex. As a result, the OptBA-TTT problem is a convex optimization problem which can be efficient solved by using convex algorithms introduced in [31].

5.4.2 Formulation of the Single Set Flow Scheduling Problem and Its Solvable Relaxation

We now study the Single Set Flow Scheduling (SSFS) problem in which we are required to determine the routing plan and bandwidth allocation plan for a set of flows with the goal of minimizing the TTT. We formulate this problem as a Non-Linear Programming (NLP) problem, present its relaxed model and further transform the relaxed model to a solvable convex optimization model.

Based on the objective and constraints introduced in Section 5.3, the SSFS problem can be formulated as

SSFS

$$Minimize \;\; TTT = \sum_{i=1}^{N} tt_i = \sum_{i=1}^{N} \frac{V_i}{b_{f_i}}. \tag{5.11}$$

Subject to

$$\sum_{w \in \mathcal{N}(s_i)} x^i_{ws_i} - \sum_{w \in \mathcal{N}(s_i)} x^i_{s_i w} = -1, \quad i = 1, \ldots, N, \tag{5.12}$$

$$\sum_{w \in \mathcal{N}(s_i)} x^i_{wd_i} - \sum_{w \in \mathcal{N}(s_i)} x^i_{d_i w} = 1, \quad i = 1, \ldots, N, \tag{5.13}$$

$$\sum_{w \in \mathcal{N}(u)} x^i_{wu} - \sum_{w \in \mathcal{N}(u)} x^i_{uw} = 0, \;\; \forall i, \forall u \notin \{s_i, d_i\}, \tag{5.14}$$

$$\sum_{i=1}^{N} \left| x^i_{uv} \cdot b_{f_i} \right| \leq B_{uv}, \;\; \forall u, \forall v, that \; E_{uv} \in \mathcal{E}, \tag{5.15}$$

$$b_{f_i} \geq 0, \;\; i = 1, \ldots, N, \tag{5.16}$$

$$x^i_{uv} \in \{-1, 0, 1\}, \;\; \forall i, \;\; \forall u, \;\; \forall v. \tag{5.17}$$

Remarks:

- The objective (5.11) and constraints (5.12)–(5.15) are formally defined in Section 5.3.
- Constraints (5.16) and (5.17) are domain constraints.

Naturally, the SSFS problem is a NLMIP problem which is hard to solve directly. To solve this problem, we first propose a solvable relaxation and

then determine a solution of the SSFS problem based on the solution of the relaxation.

5.4.2.1 A Relaxation of the SSFS problem and An Equivalent Convex Optimization Problem

To solve the SSFS problem, we first consider solving its relaxed version in which the integer variable x_{uv}^i is relaxed to real variable, i.e., substituting constraint (5.17) by

$$-1 \leq x_{uv}^i \leq 1, \ \forall i, \ \forall u, \ \forall v. \tag{5.18}$$

We name this relaxed version of SSFS as **SSFS-Relax** and present it as

SSFS-Relax

Objective:

$$Minimize \ \sum_{i=1}^{N} \frac{V_i}{b_{f_i}} \tag{5.19}$$

Constraints:

$$\sum_{w \in \mathcal{N}(s_i)} x_{ws_i}^i - \sum_{w \in \mathcal{N}(s_i)} x_{s_i w}^i = -1, \quad i = 1, \ldots, N, \tag{5.20}$$

$$\sum_{w \in \mathcal{N}(s_i)} x_{wd_i}^i - \sum_{w \in \mathcal{N}(s_i)} x_{d_i w}^i = 1, \quad i = 1, \ldots, N, \tag{5.21}$$

$$\sum_{w \in \mathcal{N}(u)} x_{wu}^i - \sum_{w \in \mathcal{N}(u)} x_{uw}^i = 0, \ \forall i, \forall u \notin \{s_i, d_i\}, \tag{5.22}$$

$$\sum_{i=1}^{N} \left| x_{uv}^{i} \cdot b_{f_i} \right| \leq B_{uv},\ \forall u, \forall v, that\ E_{uv} \in \mathcal{E}, \tag{5.23}$$

$$b_{f_i} \geq 0,\ i = 1, \ldots, N, \tag{5.24}$$

$$-1 \leq x_{uv}^{i} \leq 1,\ \forall i,\ \forall u,\ \forall v. \tag{5.25}$$

We notice that the SSFS-Relax problem is still hard to solve because of the product of two variables x_{uv}^{i} and b_{f_i} in constraint (5.14). To solve SSFS-Relax, we now transform it to an equivalent convex optimization problem named **SSFS-Relax-Cvx**. We start from defining variable p_{uv}^{i} as the product of x_{uv}^{i} and b_{f_i}, i.e.,

$$p_{uv}^{i} = x_{uv}^{i} \cdot b_{f_i}. \tag{5.26}$$

By substituting p_{uv}^{i} into each constraint of SSFS-Relax, we have

SSFS-Relax-Cvx

Objective:

$$Minimize\ \ \sum_{i=1}^{N} \frac{V_i}{b_{f_i}} \tag{5.27}$$

Constraints:

$$\sum_{w \in \mathcal{N}(s_i)} p_{ws_i}^{i} - \sum_{w \in \mathcal{N}(s_i)} p_{s_i w}^{i} = -b_{f_i},\quad i = 1, \ldots, N, \tag{5.28}$$

$$\sum_{w \in \mathcal{N}(s_i)} p_{wd_i}^{i} - \sum_{w \in \mathcal{N}(s_i)} p_{d_i w}^{i} = b_{f_i},\quad i = 1, \ldots, N, \tag{5.29}$$

$$\sum_{w \in \mathcal{N}(u)} p^i_{wu} - \sum_{w \in \mathcal{N}(u)} p^i_{uw} = 0,\ \forall i, \forall u \notin \{s_i, d_i\}, \tag{5.30}$$

$$\sum_{i=1}^{N} \left| p^i_{uv} \right| \leq B_{uv},\ \forall u, \forall v, that\ E_{uv} \in \mathcal{E}, \tag{5.31}$$

$$b_{f_i} \geq 0,\ i = 1, \ldots, N, \tag{5.32}$$

$$-b_{f_i} \leq p^i_{uv} \leq b_{f_i},\ \forall i,\ \forall u,\ \forall v. \tag{5.33}$$

We observe that objective (5.27) and constraint (5.30) are convex functions because the function $|p^i_{uv}|$ and V_i/b_{f_i} are convex and the sum of convex functions is still convex function. In addition, all other constraints are affine. Hence, the SSFS-Relax-Cvx problem is a convex optimization problem that can be solved by convex algorithms [31].

5.4.3 The Flow Routing and Bandwidth Allocation (FRoBA) Algorithm and Its Simplified Version: FRoBA-fast

To solve the SSFS problem, we propose an algorithm, called Data Flow Routing and Bandwidth Allocation with Minimizing Total Transfer Time (FRoBA). The FRoBA algorithm has three phases: First, it solves the SSFS-Relax-Cvx problem, the relaxed version of SSFS; second, it determines an initial solution of the SSFS problem based on the solution of SSFS-Relax-Cvx; third, it utilizes a local search procedure to further optimize the obtained initial solution. We now introduce the details of each phase.

Phase I: Solve the relaxed problem. In this phase, the FRoBA algorithm

solves the SSFS-Relax problem. It begins with solving the equivalent model SSFS-Relax-Cvx by using convex algorithms introduced in [31]. Assume that the solution of SSFS-Relax-Cvx is b'_{f_i} and ${p'}^{i}_{uv}$. FRoBA then calculates the value of variable x^i_{uv} based on Equation (5.26). We denote the obtained results by x'^{i}_{uv}.

Phase II: Obtain an initial solution of SSFS. In this phase, the FRoBA algorithm obtains an initial solution of SSFS from the solution of the relaxed problem SSFS-Relax. The procedure contains two steps:

- First, the FRoBA algorithm determines the value of variable x^i_{uv}, i.e., finding a route for each data flow. Specifically, for a data flow i, FRoBA uses ${x'}^{a}_{uv}$ as the capacity of link E_{uv} and find the max capacity path [81] (denoted by p^{MC_i}) between the source and destination node of flow i. Subsequently, FRoBA set x^i_{uv} as 1 for each link on the max capacity path and set its value as 0 for all other links. We can express such assignment as

$$x^i_{uv} = \begin{cases} 1, & \text{if } E_{uv} \text{ is on the path } p^{MC}_i, \\ 0, & \text{otherwise.} \end{cases} \tag{5.34}$$

- Second, the FRoBA algorithm determines the value of b_{f_i}, i.e., the amount of bandwidth allocated to each data flow. We observe that once the value of x^i_{uv} is obtained, the SSFS problem is naturally reduced to the OptBA-TTT problem. Hence, FRoBA determines the value of b_{f_i} by solving the corresponding OptBA-TTT problem reduced from the original SSFS problem.

Phase III: Local search. In this phase, the FRoBA algorithm utilizes a local search procedure to further optimize the initial solution obtained in previous phase.

In this local search procedure, the FRoBA algorithm starts from calculating the residual bandwidth of each link according to the initial solution x_{uv}^i and b_{f_i}, i.e., subtracting b_{f_i} from the available bandwidth of each link whose x_{uv}^i equals to 1.

The FRoBA algorithm then runs in iterations. In each iteration, it put all congested links (i.e., links without any residual bandwidth) into a set $\mathcal{E}_{congest}$ in the decreasing order of the number of data flows using this link. Next, FRoBA iteratively addresses each congested link in set $\mathcal{E}_{congest}$. When addressing a congested link e, the FRoBA algorithm identifies all data flows using this link and put them into set $\mathcal{F}_{congest}$ in the decreasing order of their transfer time. Subsequently, FRoBA marks the link e as unavailable and iterates through the set $\mathcal{F}_{congest}$. For a selected data flow a, FRoBA sets the available bandwidth of each link as its capacity and finds out the shortest max capacity path, i.e., the shortest path which has the largest capacity among all shortest paths. Note that the FRoBA algorithm uses the number of hops on a path as the length of that path. In the following, it temporarily changes the route of flow a to the newly found route and calculate the optimal bandwidth allocation for current routing schedule. If the total transfer time is reduced in the new solution, i.e., our solution is improved, the FRoBA algorithm then stops consider the following data flows in $\mathcal{F}_{congest}$ and congested links in $\mathcal{E}_{congest}$; instead, it starts a new iteration. Otherwise, it reverts all temporary changes that it has made and moves to the next data flow in $\mathcal{F}_{congest}$.

If the FRoBA algorithm cannot improve current solution in some iteration (after iterating through all congested links and all data flows using these links), the algorithm then finishes and outputs current solution as the final solution.

5.4.3.1 FRoBA-fast: A Simplified Version of FRoBA

The FRoBA algorithm begins with solving the SSFS-Relax-LP problem which is a LP problem. When the problem's scale is large enough, solving this problem can be time-consuming. On the other hand, we observe that the local search procedure in the FRoBA algorithm can be used to optimize any feasible schedules. With such observations, we propose FRoBA-fast, a simplified version of FRoBA with less time complexity.

FRoBA-fast has two phases: First, it obtains an initial solution; Second, it utilizes the local search procedure used in the FRoBA algorithm to optimize the initial solution. To obtain an initial solution, for each flow F_i, the FRoBA-fast algorithm set the route of this flow as the shortest maximum capacity path (i.e., the shortest path with the maximum capacity). Subsequently, FRoBA-fast calculates b_{f_i} and CCT based on the determined routes.

5.5 Online Scheduling

In the previous section, we have studied how to schedule a single set of flows with the goal of minimizing their total transfer time (TTT) and have proposed algorithms to solve this problem.

However, in practice, flows arrive the system in a time sequence and in

a long term view, our goal is to minimize the average TTT of all arrived flows. With this goal, we propose an online scheduler named OnFRoBA, which periodically schedules all arrived flows together.

5.5.1 Online Scheduler: OnFRoBA.

The OnFRoBA scheduler puts each arrived flow into a waiting queue Q_{wait} and keeps track of the waiting time of each job in Q_{wait}. The OnFRoBA scheduler periodically addresses the flows in the waiting queue and temporarily allocates residual bandwidth to ongoing flows everytime a flow finishes. In the following, we introduce details of the OnFRoBA scheduler

5.5.1.1 Periodic scheduling of the waiting queue.

The OnFRoBA scheduler periodically addresses the flows in the waiting queue in two steps:

Step I: Flow Selection. First, it selects a set of flows from Q_{wait}, denoted by $\mathcal{F}_{cand}$. Those selected flows are scheduled in the next step and those unselected flows remain in the waiting queue. The motivations of such selection include:

- First, it is possible that there is no valid route for some waiting flows. Naturally, such flows should remain waiting until there are valid routes.
- Second, there may be a large number of waiting flows. If all of them are scheduled at the same time, each of them may get only a very small portion of available bandwidth, which can largely prolong the average TTT. Under

such circumstance, scheduling some of these waiting flows and making them finished first can be a better choice.

To perform flow selection, we propose three approaches: Valid Flow First(VFF), Smallest Load First (SLF), and Shortest Flow First (SFF). The details of these approaches are introduced later.

Step II: Flow Scheduling. After selecting the flow set $\mathcal{F}_{candidate}$, the OnFRoBA scheduler calls the FRoBA algorithm to schedule the flows in the set. Note that the flow selection procedure guarantees that there is always feasible schedules for the set $\mathcal{F}_{candidate}$.

5.5.1.2 The Distribute Residual Bandwidth (DRB) procedure

When a flow is finishes, the bandwidth reserved by that flow is released and becomes residual bandwidth. For work conservation, the OnFRoBA scheduler temporarily allocates the residual bandwidth to ongoing flows. This procedure is named Distribute Residual Bandwidth (DRB).

Specifically, the DRB procedure releases the bandwidth reserved for ongoing flows and calculates the new bandwidth allocation plan, i.e., solves the OptBA-TTT problem, based on the updated bandwidth availability of the network. The DRB procedure then reserves bandwidth for ongoing flows according to the new bandwidth allocation plan.

5.5.2 Flow Selection Approaches

The details of the proposed flow selection approaches are introduced in the following.

Valid Flow First (VFF). The VFF procedure simply selects all valid flows. For a flow F_i, if there exists at least one route whose available bandwidth is larger than a certain threshold, this flow F_i is then a valid flow. To verify the feasibility of a flow F_i, the VFF procedure tries to find the maximum capacity path for F_i. If a feasible path exists, flow F_i is then valid; otherwise, it is invalid.

Shortest Flow First (SFF). The SFF procedure runs in iterations. In each iteration, it selects one flow from the waiting queue and adds it into $\mathcal{F}_{candidate}$ in three steps:

- First, the SFF procedure calculates the minimum transfer time (MinTT) of each flow in the waiting queue. Specifically, it finds out the maximum capacity path for each flow F_i. Let the available bandwidth of this path be $B_{P_i}^{max}$. The MinTT of flow F_i, denoted by $MinTT_{f_i}$ is then

$$MinTT_{f_i} = \frac{V_i}{B_{P_i}^{max}}. \tag{5.35}$$

  If such a path does not exist, the SFF procedure skips this flow and continues considering the following flow.

- Second, the SFF procedure selects the flow F_t with the shortest MinTT, i.e.,

$$F_t = \underset{F_i \in Q_{wait}}{\operatorname{argmin}} \{MinTT_{f_i}\}. \tag{5.36}$$

  The flow F_t is then moved from Q_{wait} to $\mathcal{F}_{candidate}$.

- Third, the SFF procedure temporarily updates the available bandwidth of

the links on the maximum capacity path of F_t, i.e., reducing the amount of available bandwidth by $B_{P_t}^{max}$.

After performing the three steps, the SFF procedure starts the next iteration. If in some iteration, there is no flow that has a valid path, the SFF procedure stops the selection and reverses all updates made on the available bandwidth during the iterations. The intuition of the SFF procedure is that if some flows compete for the available bandwidth on some links, we may be able to get smaller TTT by transferring flows with shorter transfer time first. This is similar to the well-known Shortest Job First approach used in task scheduling.

Smallest Load First (SLF). The SLF procedure iterates through flows in the waiting queue in the ascending order of their volume. For each flow F_i, the SLF procedure determines the maximum capacity path. If a valid path with available bandwidth $B_{P_i}^{max}$ exists, the SLF procedure adds F_i into $\mathcal{F}_{candidate}$ and temporarily reserve the bandwidth $B_{P_i}^{max}$ along the path; otherwise, the SLF procedure directly moves to the next flow. After addressing each flow in Q_{wait}, the SLF procedure stops selection and reverses all updates made on the available bandwidth.

The SLF procedure follows the similar principle used in the SFF procedure but reduces the complexity by selecting flows in the ascending order of their volume. The intuition here is that it is more likely that a flow with smaller volume has a shorter transfer time.

5.5.3 Preemptive Flows

It is possible that flows are preemptive in some data transferring systems. If flows are preemptive, before selecting the flow set $F_{candidate}$ from the waiting queue, the OnFRoBA scheduler preempts all ongoing flows and adds them into the waiting queue. Subsequently, it generates $F_{candidate}$ and call scheduling algorithm based on all existing flows. Besides this major change, whenever the volume of a flow is considered in any scheduling procedure, the residual volume of this flow is considered.

Moreover, when flows are preemptive, the DRB procedure allocates residual bandwidth to waiting flows also. When distributing the residual bandwidth to waiting flows, the DRB procedure iterates through the flows in Q_{wait} in the ascending order of their volume. For each flow F_i, the DRB procedure finds out the maximum capacity path. If such path exists, assuming that its available bandwidth is $B_{P_i}^{max}$, the DRB procedure then allocates $B_{P_i}^{max}$ amount of bandwidth to F_i along the maximum capacity path and updates the residual bandwidth accordingly. If a maximum capacity path does not exist, the DRB procedure moves to the next flow. After addressing each flow in Q_{wait}, the DRB procedure finishes

5.6 Performance Evaluation

5.6.1 Performance of the FRoBA Algorithm

In this section, we evaluate the FRoBA algorithm through offline simulations. In the following, we present our simulation setup, evaluation metrics,

comparing algorithms and simulation results.

5.6.1.1 Simulation Setup

In each single run of the simulation, we randomly generate a data center network and a set of input flows. The FRoBA algorithm is then called to schedule the flows on the given network.

Flow Set. We randomly generate a set of N flows. For each flow, we randomly select two hosts as its source and destination. We further set the maximum possible volume of a flow (denoted by V_{max}) as 1000 Gb and determines the volume of a flow (i.e., V_i) by using an uniform distribution in $[\beta \cdot V_{max}, V_{max}]$. In our simulations, we set β as 0.7.

Data center network. For the network, we use a modified FatTree [64] architecture. A k-array FatTree network has k pods, where each pod has $k/2$ Top-of-Rack (ToR) switches and $k/2$ aggregation switches. While in each pod the ToR and aggregation switches are interconnected as a complete bipartite graph, each ToR switch also connects a rack of $k/2$ hosts. In addition, there are $(k/2)^2$ core switches that connect the aggregation switches of all pods. In general, a k-array FatTree network is able to support $k^3/4$ hosts.

To better evaluate our algorithms, we decide to increase the oversubscription ratio and therefore introduce more competition on bandwidth at the aggregation and core levels. To achieve this goal, we multiply the number of hosts in a rack by a factor α_{over}. By doing so, a k-array modified FatTree contains $\alpha_{over} \cdot k^3/4$ hosts now. In our simulations, we set α_{over} as 2 and set the each link's capacity as 10 Gbps.

Noise flows. In order to simulate the complex traffic condition in real data

center, we introduce noise flows into our simulations. Specifically, each single simulation run starts with generating a set of noise flows whose amount is 4 times of the number of hosts in the network. The source and destination of each noise flow is randomly selected and the duration follows an uniform distribution in $[1, 150]$. We randomly select a shortest path between as its route and allocate a certain amount of bandwidth which is the available bandwidth of the selected path multiplied by a random factor within $[0, 0.5]$.

5.6.1.2 Evaluation Metrics

We use four evaluation metrics to evaluate our algorithms.

Total Transfer Time. This metric is the objective of the CoS problem. Hence, it is the most important metric.

Allocated bandwidth. This metric is the overall bandwidth allocated to all flows in $\mathcal{F}$, which equals to $\sum_{i=1}^{N} b_{f_i}$. It demonstrates how an algorithm performs on the aspect of bandwidth allocation.

Average length of flow route. This metric is the average number of hops on the route of flows in $\mathcal{F}$. It gives us a sense about how an algorithm performs on the aspect of routing.

Running time. Running time of an algorithm is also important. It gives a sense of the scalability of that algorithm.

5.6.1.3 Comparison Algorithms

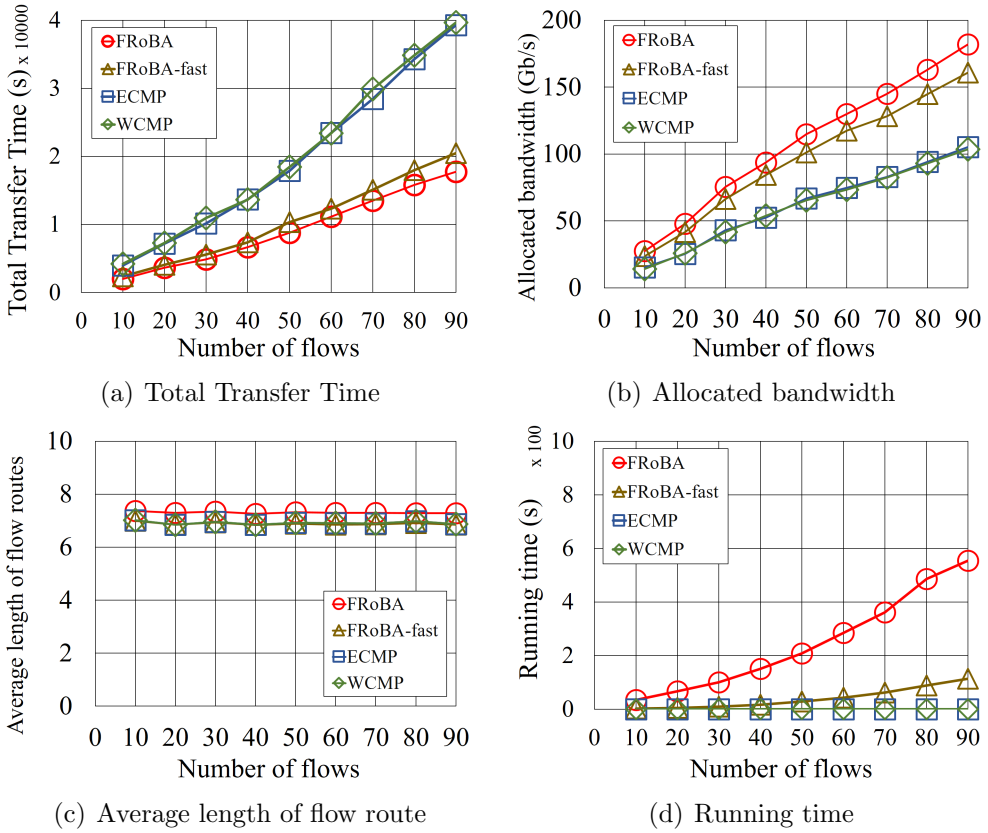
We compare our algorithms with two other algorithms.

Equal Cost Multipath Forwarding (ECMP). The ECMP algorithm is one of the most popular routing algorithm using in data centers in nowa-

days, because of its simplicity and fast running speed. The ECMP algorithm randomly selects a shortest path between the source and destination of each flow. Because the ECMP algorithm does not perform any bandwidth allocation, in practice, some bandwidth allocation approach should be called to allocate bandwidth to the flows according to the selected routes. In the simulations, we solves the OptBA-TTT problem to allocated bandwidth based on the routing plan determined by the ECMP algorithm.

Weighted Cost Multipath Forwarding (WCMP). The WCMP algorithm is a variant of the ECMP algorithm. For a given flow, the WCMP algorithm gives each shortest path between the source and destination a certain weight and then randomly selects one path based on the assigned weights. In the simulations, the WCMP algorithm uses the available bandwidth of each shortest path as its weight and allocated bandwidth by solving the OptBA-TTT problem. Generally, the WCMP algorithm should have better performance on load balance and should have larger chance to allocate more bandwidth to a given flow, compared to the ECMP algorithm.

Note that the ECMP and WCMP algorithms used in our simulations optimally allocate bandwidth by using the solution proposed in our paper. However, these two algorithm perform routing and bandwidth allocation in two totally separately steps. By comparing our algorithms with them, we essentially demonstrate the importance of jointly considering the routing and bandwidth allocation.



(a) Total Transfer Time

(b) Allocated bandwidth

(c) Average length of flow route

(d) Running time

Figure 5.1: Performance of FRoBA and FRoBA-fast when scheduling different numbers of flows in a FatTree network with 1617 nodes.

5.6.1.4 Evaluation Results of FRoBA and FRoBA-fast

Performance with Increasing Number of Flows. In this simulation, we study how FRoBA and FRoBA-fast perform as the number of flows in the coflow increases from 10 to 90. We uses a 16-array modified FatTree with $\alpha_{over} = 2$, which contains 1617 nodes.

Fig. 5.1(a) shows the TTT generated by each algorithm. While the TTT generally increases along with the expansion of the coflow, FRoBA generates

the smallest CCT. When the coflow contains 80 flows, the TTT of FRoBA is 120% smaller than that of ECMP and WCMP, and 15% smaller than that of FRoBA-fast. We also observe that the growth rate of the TTT of FRoBA and FRoBA-fast is much lower than that of ECMP and WCMP. When the number of flows increases from 10 to 90, the growth rate of the TTT generated by FRoBA and FRoBA-fast is round 200 second/flow, but that of ECMP and WCMP is about 400 seconds/flow.

Fig. 5.1(b) shows the total bandwidth allocated to the coflow. As can be seen, FRoBA allocates about 70%-100% more bandwidth than the ECMP and WCMP algorithms and about 10%-15% more bandwidth than FRoBA-fast. Fig. 5.1(c) shows the average length of flow route. We observe that the FRoBA algorithm generates the longest flow route length, followed by FRoBA-fast, ECMP, and WCMP.

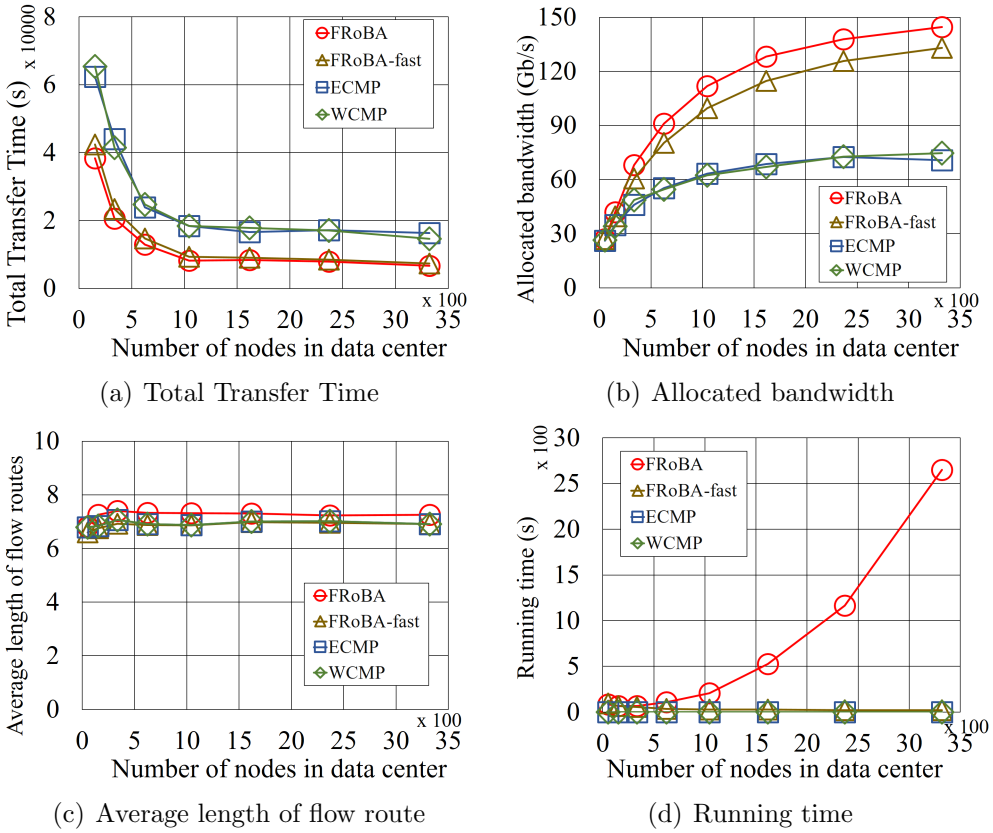
Putting Figs. 5.1(a), 5.1(b) and 5.1(c) together, we can see that when the number of the input flows increases, FRoBA and FRoBA-fast are able to allocate more bandwidth to the coflow and thereby keep the growth of TTT relatively slow, as observed in Fig. 5.1(a). We attribute this to their ability of routing based on the bandwidth availability of the whole network. When the number of flows increases, FRoBA and FRoBA-fast are able to route flows via different paths and utilize more bandwidth. In contrast, the ECMP and WCMP algorithms select the routes for input flows separately and randomly. With such selection, the generated routes may share a lot of links and may also use links with poor bandwidth availability. Consequently, these algorithms cannot sufficiently utilize the available resources in the network, which leads the rapid increase of the TTT.

At last, Fig. 5.1(d) shows the algorithm running time. When the coflow contains 90 flows, the running time of FRoBA is about 550 seconds, while that of FRoBA-fast, ECMP, and WCMP is 113 seconds, 1 seconds, and 1 seconds respectively. One of the reasons is that FRoBA solves a programming problem containing a large number of variables, which is commonly a slow procedure. Another reason is the hardware limitation: All simulations are perform on a laptop with Intel i5-3250M CPU with 2.5 GHz speed. We believe that the algorithm can run much faster on servers with better CPUs. In addition, parallel convex optimization techniques have been well studied recently [82–84]. By utilizing such techniques, the FRoBA algorithm can be executed in a parallel environment and its running speed can be further improved.

Performance with Increasing Size of the Network. In this simulation, we demonstrate how our algorithms perform as the number of nodes in the FatTree network increases from 153 to 3300, i.e., the number of pods increases from 6 to 18. We fix the number of flows at 50.

Fig. 5.2(a) shows the TTT. We can see that the TTT of FRoBA is about 10%-15% smaller than that of FRoBA-fast, 60%-250% smaller than that of ECMP and WCMP Fig. 5.2(b) shows the total allocated bandwidth. FRoBA allocates 13% more bandwidth than FRoBA-fast and about 20%-100% more bandwidth than ECMP and WCMP.

As can be seen, the bandwidth allocated by FRoBA is dramatically increased when the network just starts expanding and becomes more smooth thereafter. We attribute this to the limitation on how much bandwidth the algorithm can find to utilize. When the network is small, the number of good



(a) Total Transfer Time

(b) Allocated bandwidth

(c) Average length of flow route

(d) Running time

Figure 5.2: Performance of FRoBA and FRoBA-fast when scheduling 50 flows in FatTree networks with different number of nodes.

paths (with large available bandwidth) is also small. Consequently, FRoBA may schedule some flows to use paths with less bandwidth, which limits the TTT. At this stage, an expansion of network generates more good paths and make FRoBA be able to allocate more bandwidth. However, along with the expansion, the number of good paths becomes more than enough and FRoBA is able to schedule most of the flows on these paths. At this stage, the quality of paths is barely improved. As a result, the increase of total allocated bandwidth becomes small. Such a trend on the allocated bandwidth

then leads the trend of TTT, which is a steep decrease at the beginning but becomes relatively flat thereafter, as shown in Fig. 5.2(a).

We also observe that although the ECMP and WCMP algorithms optimally allocate bandwidth (by solving the OptBA problem), they actually allocate much less bandwidth than FRoBA. This is because these algorithms perform routing and bandwidth allocation separately, which may cause the selected routes sharing a lot of links and using links with low bandwidth availability. As a result, even if they optimally allocate bandwidth based on the selected routes, the total amount of allocated bandwidth is small, which then leads a large value of TTT. Whereas FRoBA takes the whole network into consideration when performing routing and thereby is able to find routes with less or even no overlap and allocate more bandwidth.

Fig. 5.2(c) shows the average length of flow routes. While the FRoBA algorithm generates the longest routes, the average lengths of flow routes generated by the four algorithms are actually close to each other.

At last, Fig. 5.2(d) shows the algorithm running time. When the network contains 16 pods (i.e., 4500 nodes), the running time of FRoBA is about 2600 seconds, while that of FRoBA-fast,ECMP, and WCMP is 20 seconds, 1 second, and 1 second respectively.

5.6.1.5 Summary

In offline simulations, we examined the performance of FRoBA and FRoBA-fast when scheduling a single set of input flows. Benefiting from the ability of finding paths with less overlaps and allocating more bandwidth, FRoBA and FRoBA-fast outperforms outperforms ECMP and WCMP by 60%-250%.

5.6.2 Performance of the OnFRoBA Scheduler

In this section, we examine the performance of the OnFRoBA scheduler through online simulations.

5.6.2.1 Simulation Setup

An online simulation begins with a FatTree network without any flows. In the following, flows start to arrive at a rate following a Poisson distribute with $\mu = 0.8$/second, and stop arriving at 1200 seconds. The whole simulation finishes, once all flows are scheduled. Each data point in the results is an average of 10 simulations performed on an Intel 2.5GHz processor.

Flows. Flows start to arrive at a rate following a Poisson distribute with $\mu = 0.8$/second, and stop arriving at 1200 seconds. For each flow, we randomly select two hosts as its source and destination. We further set the maximum possible volume of a flow (denoted by V_{max}) as 1000 Gb and determines the volume of a flow (i.e., V_i) by using an uniform distribution in $[\beta \cdot V_{max}, V_{max}]$. In our simulations, we set β as 0.7.

Data Center Network. We use a 10-array modified FatTree network which contains 625 nodes. The capacity of links in the network is set to 10 Gbps.

Noise flows. The noise flows arrives following a Poisson distribution and the duration of these noise flows follows an uniform distribution in $[1, 150]$. When a noise flow starts, we randomly selects a shortest path as its route and allocates a certain amount of bandwidth which is the link capacity times a random factor within $[0, 0.5]$.

5.6.2.2 Evaluation Metrics

We use two metrics to evaluate the OnFRoBA scheduler.

Average Flow Completion Time (FCT). The FCT of flow F_i, denoted by FCT_{f_i}, is the length of the time period between the arriving time and the completion time of that flow. It composed of the total waiting time of that flow (denoted by FWT_{f_i}) and the flow transfer time of that flow (denoted by FTT_{f_i}) generated by the scheduling algorithm, i.e.,

$$FCT_{f_i} = FWT_{f_i} + FTT_{f_i}. \tag{5.37}$$

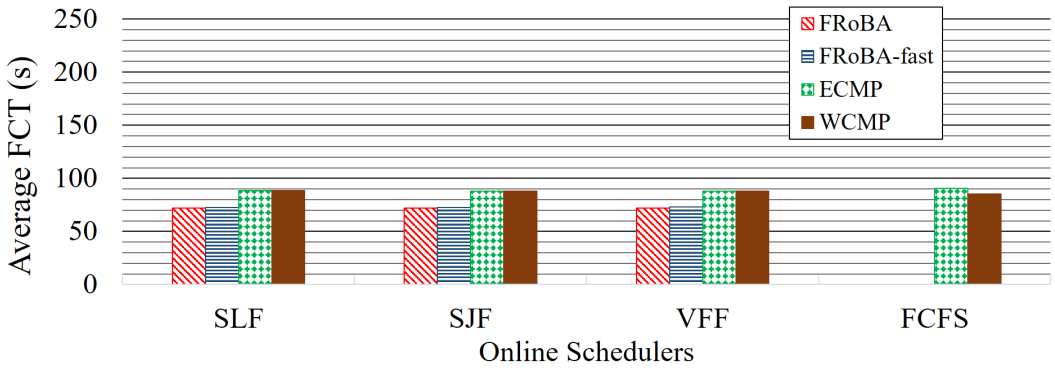
The average FCT is the average of FCT of all arrived flows.

Total Running Time. Total running time is the running time of a scheduler from the beginning of a simulation to the end.

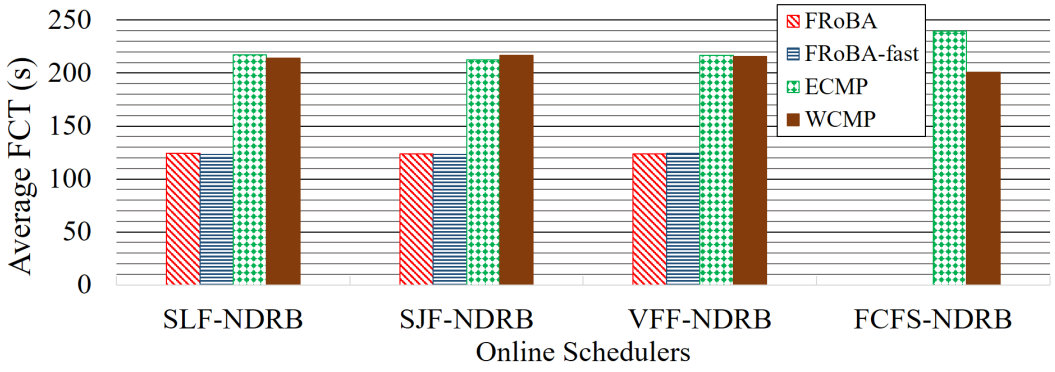
5.6.2.3 Comparison Scheduling Policies and Algorithms

In the online simulations, we compare the three flow selection approaches proposed in our paper. For each of these approaches, we further compare the FRoBA and FRoBA-fast algorithm with the ECMP and WCMP algorithms.

FCFS. Furthermore, we compare the proposed scheduler with a FCFS scheduler. The FCFS scheduler addresses the flows in the waiting queue whenever a new flow arrives. Specifically, it iterates the flows in the waiting queue in the ascending order of their arriving time, following the principle of first come first serve. For each flow, the FCFS scheduler calls the ECMP algorithm or the WCMP algorithm to schedule that flow. If a flow fails to be scheduled, it is added into waiting queue and wait for the next round of scheduling.



(a) DRB Enabled

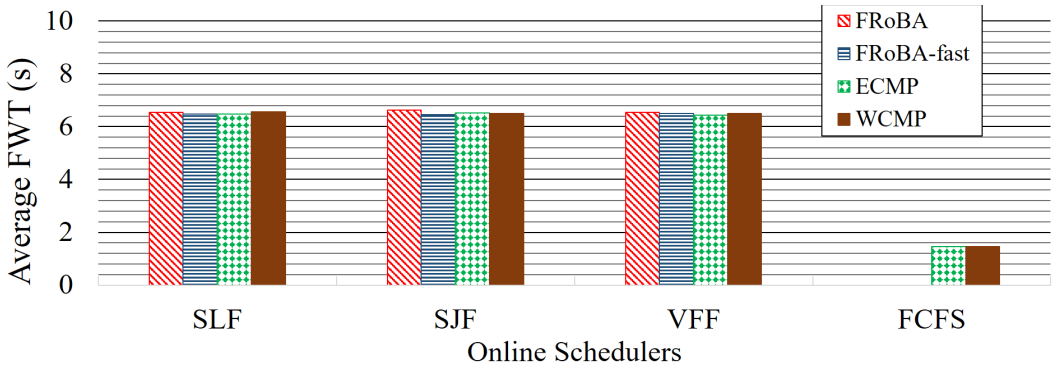


(b) DRB Disabled

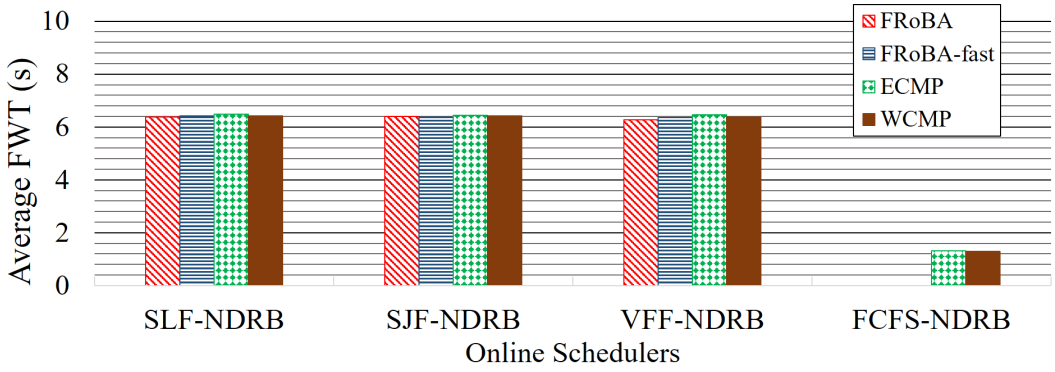
Figure 5.3: The average FCT generated by different scheduler when flows are non-preemptive.

5.6.2.4 Performance of OnFRoBA when Flows are Non-preemptive

Fig. 5.3 shows the average FCT generated by each combination of flow selection approaches and scheduling algorithms with/without the DRB procedure enabled. Fig. 5.4 shows the average FWT generated by each combination of flow selection approaches and scheduling algorithms with/without the DRB procedure enabled. Fig. 5.5 shows the running time of each combination. Based on the results, we have several observations:



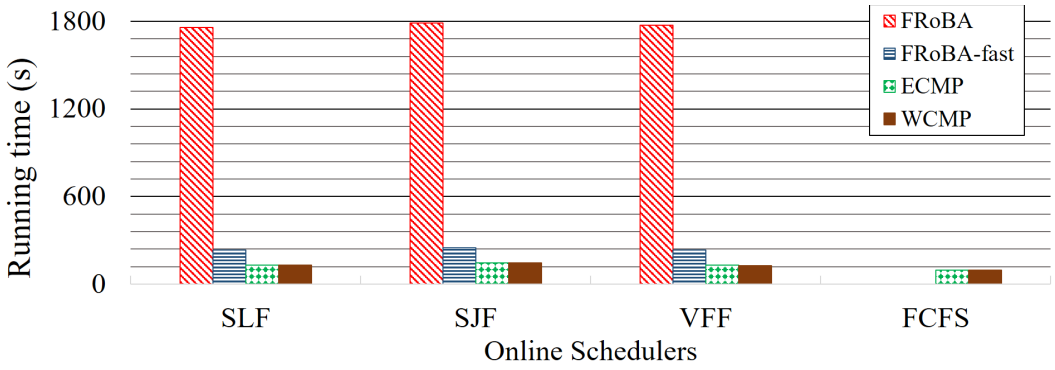
(a) DRB Enabled



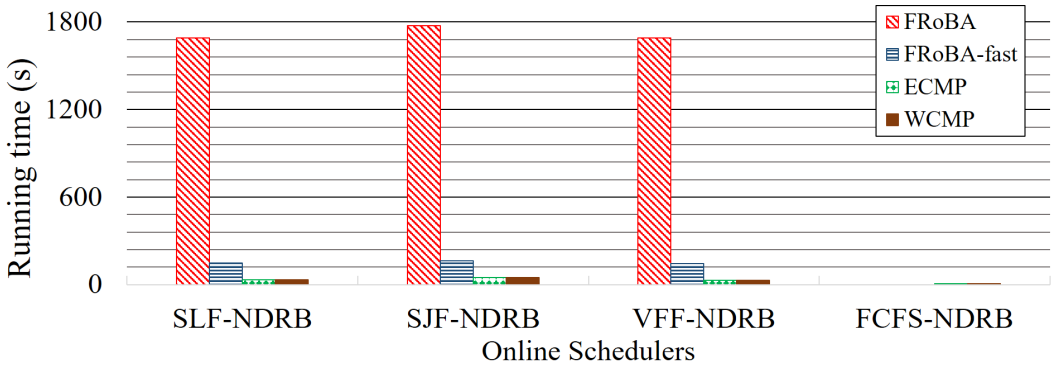
(b) DRB Disabled

Figure 5.4: The average FWT generated by different scheduler when flows are non-preemptive.

Impact of Scheduling Algorithms. As shown in Fig. 5.3, FRoBA and FRoBA-fast generally show better performance than ECMP and WCMP, while FRoBA is about 5% better than FRoBA-fast. As shown in Fig. 5.3(a), in the DRB enabled simulations, FRoBA and FRoBA-fast reduces the average FCT by about 20% compared to ECMP and WCMP in the SLF, SJF, and VFF selection respectively. Furthermore, compared to the FCFS selection, FRoBA and FRoBA-fast in the SLF selection reduces the average FCT



(a) DRB Enabled



(b) DRB Disabled

Figure 5.5: The running time of different scheduler when flows are non-preemptive.

by about 25%. As shown in Fig. 5.3(b), in the DRB disabled simulations, FRoBA and FRoBA-fast still shows better performance compared by ECMP and WCMP. Generally, FRoBA and FRoBA-fast performs 70%-90% better than ECMP and WCMP in all combinations with flow selection approaches.

From Fig. 5.4, we can see that the four scheduling algorithms generates similar FWT in all combinations with flow selection approaches.

Impact of the DRB procedure. The DRB procedure temporarily dis-

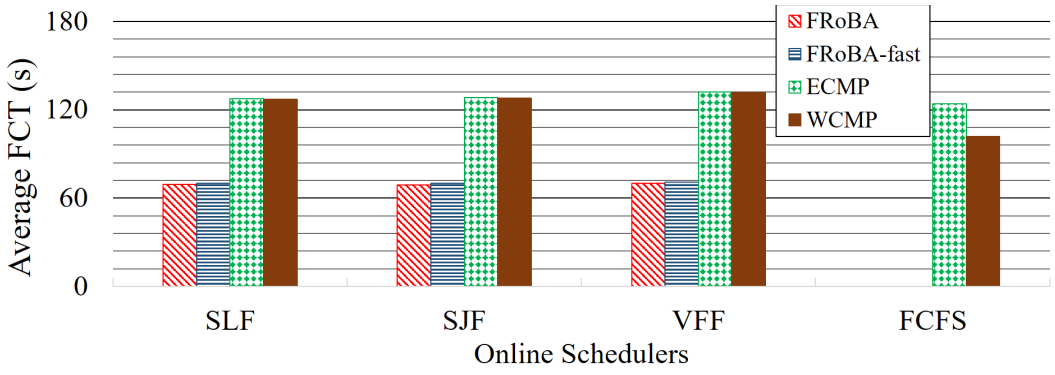
tributes the residual bandwidth in the network to the existing flows to accelerate the completion of those flows. We can see that the DRB procedure has signification impact on the average FCT. With the DRB procedure enabled, the average FCT generated by FRoBA and FRoBA-fast is reduced by about 40%, while the average FCT generated by ECMP and WCMP reduced by about 60%.

Scheduler Running Time. Fig. 5.5 shows the running time of the scheduler with each combination of flow selection approaches and scheduling algorithms. We can see that the FRoBA algorithm generally has the largest running time, followed by FRoBA-fast, ECMP, and WCMP. Meanwhile, the running time of the schedulers with DRC enabled is longer than the schedulers with DRC disabled. Furthermore, we observe that the running time of the scheduler with the SJF selection increases insignificantly compared to schedulers with SLF or VFF selection.

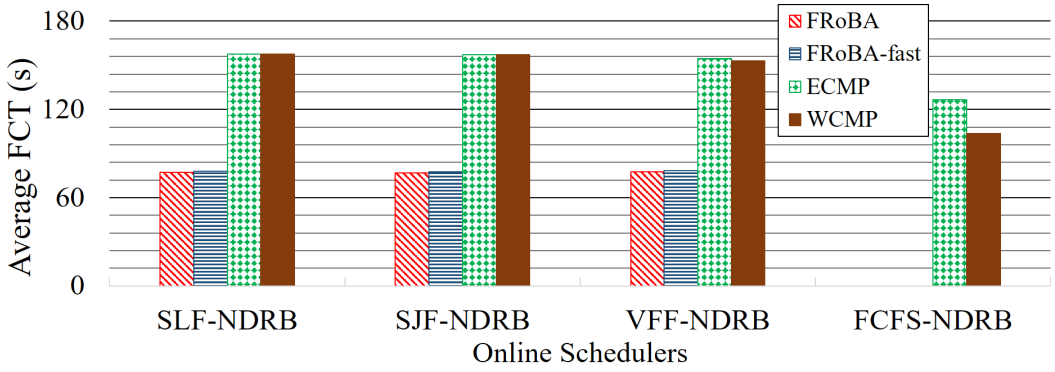
5.6.2.5 Performance of OnFRoBA when Flows are Preemptive

We now demonstrate the performance of the OnFRoBA scheduler when flows are preemptive. Fig. 5.6 and Fig. 5.7 shows the average FCT and FWT generated by each combination of flow selection approaches and scheduling algorithms with/without the DRB procedure enabled respectively. Fig. 5.8 shows the running time of each combination. Based on the results, we have several observations:

Impact of Scheduling Algorithms. As shown in Fig. 5.6, FRoBA and FRoBA-fast generally show better performance than ECMP and WCMP, while FRoBA is about 5% better than FRoBA-fast. As shown in Fig. 5.6(a),



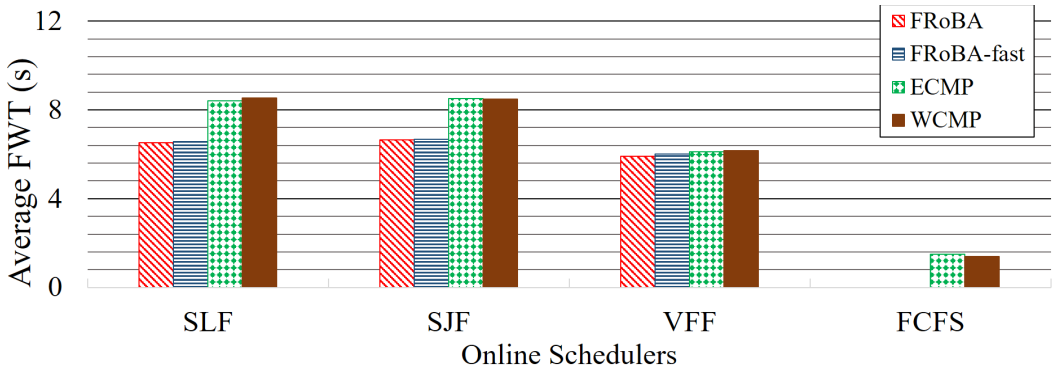
(a) DRB Enabled



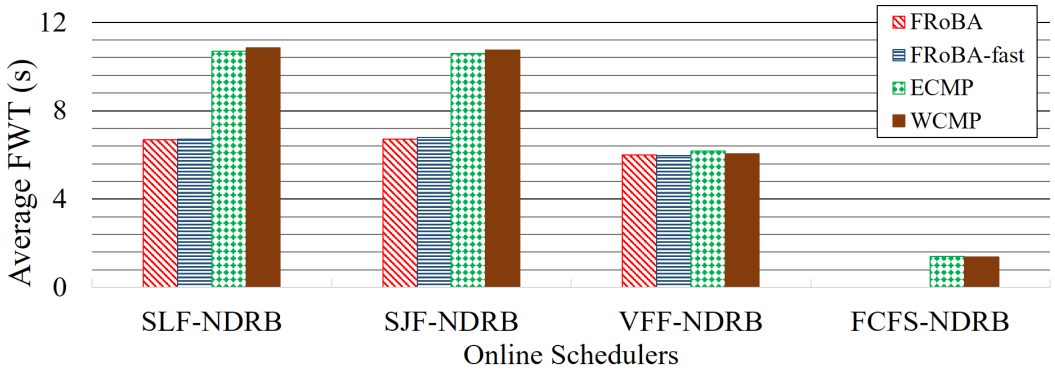
(b) DRB Disabled

Figure 5.6: The average FCT generated by different scheduler when flows are preemptive.

in the DRB enabled simulations, FRoBA and FRoBA-fast reduces the average FCT by about 85% compared to ECMP and WCMP in the SLF, SJF, and VFF selection. Furthermore, compared to the FCFS selection, FRoBA and FRoBA-fast in the SLF selection reduces the average FCT by about 50-80%. As shown in Fig. 5.6(b), in the DRB disabled simulations, FRoBA and FRoBA-fast still shows better performance compared to the ECMP and WCMP algorithms. Generally, FRoBA and FRoBA-fast performs 60%-100%



(a) DRB Enabled

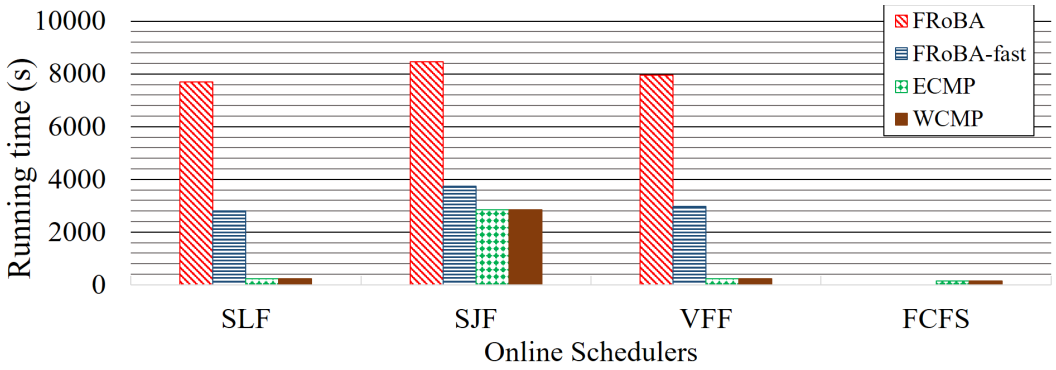


(b) DRB Disabled

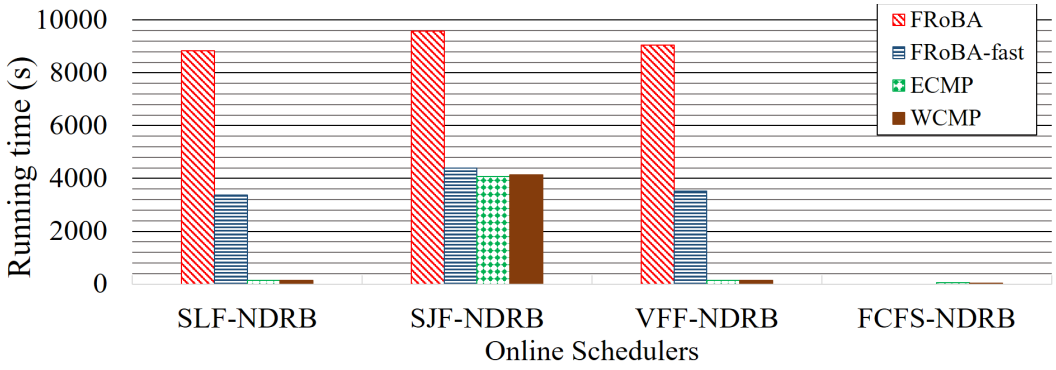
Figure 5.7: The average FWT generated by different scheduler when flows are preemptive.

better than ECMP and WCMP in all combinations with flow selection approaches.

From Fig. 5.7, we can see that FRoBA and FRoBA-fast generates smaller FWT than ECMP and WCMP when the SFL or SJF selection is used. However, the ECMP and WCMP algorithms with the FCFS selection generates the smallest FWT. This is reasonable because with the FCFS selection, the OnFRoBA scheduler attempts to schedule a flow as soon as the flow arrives.



(a) DRB Enabled



(b) DRB Disabled

Figure 5.8: The running time of different scheduler when flows are preemptive.

Impact of the DRB procedure. The DRB procedure temporarily distributes the residual bandwidth in the network to the existing flows to accelerate the completion of those flows. From Fig. 5.6(b) and Fig. 5.6(a), we can see that the DRB procedure improves the performance of the FRoBA and FRoBA-fast algorithms by 10%, while it improves the performance of ECMP and WCMP by about 20%. In addition, from Fig. 5.7, we observe that the DRB procedure also reduces the waiting time generated by ECMP

and WCMP.

Scheduler Running Time. Fig. 5.5 shows the running time of the scheduler with each combination of flow selection approaches and scheduling algorithms. We can see that the FRoBA algorithm generally has the largest running time, followed by FRoBA-fast, ECMP, and WCMP. Meanwhile, the running time of the schedulers with DRC enabled is longer than the schedulers with DRC disabled. Moreover, we observe that the running time of ECMP and WCMP with SJF selection is much higher than with other selection approaches. We attribute to the fact that the complexity of the SJF selection is higher than other selection approaches.

5.6.2.6 Summary

In online simulations, we demonstrate the performance of the OnFRoBA scheduler using different combinations of flow selection approaches and scheduling algorithms. The results show that: (a) Compared by ECMP and WCMP, the FRoBA algorithm reduces the average FCT by 20%-25% when flows are non-preemptive and by 50%-80% when flows are preemptive; (b) The performance of the proposed flow selection approaches do not have significant difference; (c) The DRB procedure can effectively reduce the average FCT by 20%-40% when flows are non-preemptive and by 10%-20% when flows are preemptive; (d) While FRoBA-fast performs 5% worse than FRoBA, when using FRoBA-fast, the running time of OnFRoBA can be 3-14 times smaller than using FRoBA, which makes OnFRoBA+FRoBA-fast an very applicable choice in practice.

5.7 Conclusion

In this paper, we focused on the flow scheduling problem in which there is a need to perform routing and bandwidth allocation for a given set of flows with the goal of minimizing the Total Transfer Time (TTT). We first studied how to optimally allocate bandwidth to a set of flows with pre-determined routes. We formulated this problem as a convex optimization problem that can be solved efficiently. Subsequently, we formulated the Single Set Flow Scheduling (SSFS) problem as a MINLP problem and presented a relaxation of this problem together with an equivalent convex optimization problem. We further propose an algorithm named FRoBA and its simplified version: FRoBA-fast that solve the SSFS problem. At last, to address multiple flows in online scheduling, we propose an online scheduler named OnFRoBA.

We evaluated the performance of the FRoBA and FRoBA-fast algorithms and the OnFRoBA scheduler by comparing them with the state-of-the-art algorithms and schedulers in offline and online simulations. The simulation results show that: (a) FRoBA and FRoBA-fast reduce the JCT by 60%-250% compared to the state-of-the-art algorithms. (b) The OnFRoBA scheduler when using FRoBA/FRoBA-fast reduces the average FCT by 20%-80% compared to exsiting schedulers. (c) FRoBA-fast can be 10 times faster than FRoBA with around 5% performance degradation compared to FRoBA, which makes the use of FRoBA-fast very applicable in practice.

Chapter 6

Future Work

In this section, we discuss about our future work which include two aspects: (a) Extension of current study; and (b) More types of tasks.

6.1 Extension of Current Study

While we have studied three complex task scheduling problems in this dissertation, we can further extend our current study on these problems. In our study on the coflow scheduling problem, we have not considered online scheduling. Instead, we compared our scheduling algorithm with the state-of-the-art algorithms using an existing online scheduler. Based on our study and understanding of the coflow scheduling problem, how can we propose a better online scheduler is a question waiting for us to answer.

Meanwhile, these scheduling problems can be even more complex in practice, when considering more practical constraints. One important constraint is the deadline of user tasks, which has not been considered in our study

yet. Deadline-aware task scheduling is an important part of the general task scheduling problem. Furthermore, energy-aware flow scheduling is another practical problem which attracts more and more attention recently, as the energy consumed by network device is rapidly increasing. This problem is also included in our future work.

6.2 More Types of Tasks.

In this dissertation, we mainly considered embarrassingly parallel jobs, coflows and independent flows. There are many other types of tasks executing in data centers in nowadays. How to jointly perform task placement and resource allocation for those tasks is another important part of our future work.

For example, scientific workflows presents a large part of parallel processing jobs. These workflows are composed of staged computing tasks and data movements between these tasks. They are usually presented as Directed Acyclic Graphs (DAGs). Scheduling these workflows involves scheduling both computing tasks and data transfers, which is even more complex than the problems studied in our dissertation. Another example is MapReduce-like distributed data processing jobs. Such jobs also include computing tasks and data transfers, however, have strong fixed execution pattern usually. How to design specific scheduling strategies for this type of jobs is also an important problem to solve in our future work.

Chapter 7

Conclusion

In this dissertation, we studied the problem of task scheduling in data centers, which includes task placement and resource allocation. We started from a fundamental problem: how to optimally allocate resource according to determined task placements. We formulated this problem as a convex optimization problem with generalized linear constraints and presented two variants with two different but common objectives. Based on the solution of this problem, we further studied three more complex problems: (a) Energy-aware scheduling of embarrassingly parallel jobs and resource allocation in a cloud, in which there is a need to determine the task placement plan and the resource allocation plan for jobs composed of independent tasks with the goal of minimizing the Job Completion Time (JCT); (b) Coflow scheduling in data centers: routing and bandwidth allocation, in which there is a need to perform routing and bandwidth allocation for a given coflow with the goal of minimizing the coflow completion time. (c) Scheduling of independent flows in data centers: routing and bandwidth allocation, in which there is

a need to perform routing and bandwidth allocation for a given set of flows with the goal of minimizing the Total Transfer Time (TTT). We formulate each of these problems as a Non-linear Mixed Integer Programming problem and presented an relaxation with equivalent solvable problem. We further proposed offline algorithms and online schedulers, which jointly consider task placement and resource allocation, to solve these problems. Lastly, we compared the proposed solutions with existing approaches through simulations and demonstrated the superior performance of the proposed solutions.

Bibliography

[1] Y. Jiang, "A survey of task allocation and load balancing in distributed systems," *Parallel and Distributed Systems, IEEE Transactions on*, vol. PP, no. 99, pp. 1–1, 2015.

[2] T. D. Braun, H. J. Siegel, N. Beck, L. L. Blni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, B. Yao, D. Hensgen, and R. F. Freund, "A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems," *Journal of Parallel and Distributed Computing*, vol. 61, no. 6, pp. 810 – 837, 2001.

[3] X. He, X. Sun, and G. Von Laszewski, "Qos guided min-min heuristic for grid task scheduling," *Journal of Computer Science and Technology*, vol. 18, no. 4, pp. 442–451, 2003.

[4] K. Etminani and M. Naghibzadeh, "A min-min max-min selective algorihtm for grid task scheduling," in *Internet, 2007. ICI 2007. 3rd IEEE/IFIP International Conference in Central Asia on.* IEEE, 2007, pp. 1–7.

[5] J. Xu, A. Lam, and V. Li, "Chemical reaction optimization for task scheduling in grid computing," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 22, no. 10, pp. 1624–1631, Oct 2011.

[6] T. Kokilavani and D. D. G. Amalarethinam, "Load balanced min-min algorithm for static meta-task scheduling in grid computing," *International Journal of Computer Applications*, vol. 20, no. 2, pp. 43–49, 2011.

[7] E. Kartal Tabak, B. Barla Cambazoglu, and C. Aykanat, "Improving the performance of independenttask assignment heuristics minmin, maxmin and sufferage," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 25, no. 5, pp. 1244–1256, 2014.

[8] S. U. Khan and I. Ahmad, "A cooperative game theoretical technique for joint optimization of energy consumption and response time in computational grids," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 20, no. 3, pp. 346–360, 2009.

[9] J. Kołodziej, S. U. Khan, L. Wang, A. Byrski, N. Min-Allah, and S. A. Madani, "Hierarchical genetic-based grid scheduling with energy optimization," *Cluster Computing*, vol. 16, no. 3, pp. 591–609, 2013.

[10] F. Pinel, B. Dorronsoro, J. E. Pecero, P. Bouvry, and S. U. Khan, "A two-phase heuristic for the energy-efficient scheduling of independent tasks on computational grids," *Cluster computing*, vol. 16, no. 3, pp. 421–433, 2013.

[11] K. Li, X. Tang, and K. Li, "Energy-efficient stochastic task scheduling on heterogeneous computing systems," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 25, no. 11, pp. 2867–2876, 2014.

[12] D. Li and J. Wu, "Minimizing energy consumption for frame-based tasks on heterogeneous multiprocessor platforms," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 26, no. 3, pp. 810–823, March 2015.

[13] C. Guo, G. Lu, H. J. Wang, S. Yang, C. Kong, P. Sun, W. Wu, and Y. Zhang, "Secondnet: a data center network virtualization architecture with bandwidth guarantees," in *Proceedings of the 6th International COnference*. ACM, 2010, p. 15.

[14] A. Shieh, S. Kandula, A. Greenberg, C. Kim, and B. Saha, "Sharing the data center network," in *Proceedings of the 8th USENIX conference on Networked systems design and implementation*. USENIX Association, 2011, pp. 23–23.

[15] L. Popa, G. Kumar, M. Chowdhury, A. Krishnamurthy, S. Ratnasamy, and I. Stoica, "Faircloud: sharing the network in cloud computing," in *Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication*. ACM, 2012, pp. 187–198.

[16] L. Popa, P. Yalagandula, S. Banerjee, J. C. Mogul, Y. Turner, and J. R. Santos, "Elasticswitch: practical work-conserving bandwidth guaran-

tees for cloud computing," in *Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM*. ACM, 2013, pp. 351–362.

[17] H. Ballani, K. Jang, T. Karagiannis, C. Kim, D. Gunawardena, and G. O'Shea, "Chatty tenants and the cloud network sharing problem." in *NSDI*, 2013, pp. 171–184.

[18] U. Bhoi and P. N. Ramanuj, "Enhanced max-min task scheduling algorithm in cloud computing," *International Journal of Application or Innovation in Engineering and Management (IJAIEM)*, pp. 259–264, 2013.

[19] G. Ming and H. Li, "An improved algorithm based on max-min for cloud task scheduling," in *Recent Advances in Computer Science and Information Engineering*. Springer, 2012, pp. 217–223.

[20] H. Chen, F. Wang, N. Helian, and G. Akanmu, "User-priority guided min-min scheduling algorithm for load balancing in cloud computing," in *Parallel Computing Technologies (PARCOMPTECH), 2013 National Conference on*, Feb 2013, pp. 1–8.

[21] K. H. Kim, A. Beloglazov, and R. Buyya, "Power-aware provisioning of cloud resources for real-time services," in *Proceedings of the 7th International Workshop on Middleware for Grids, Clouds and e-Science*. ACM, 2009, p. 1.

[22] C.-M. Wu, R.-S. Chang, and H.-Y. Chan, "A green energy-efficient scheduling algorithm using the dvfs technique for cloud datacenters," *Future Generation Computer Systems*, vol. 37, pp. 141–147, 2014.

[23] S. Hosseinimotlagh, F. Khunjush, and R. Samadzadeh, "Seats: smart energy-aware task scheduling in real-time cloud computing," *The Journal of Supercomputing*, vol. 71, no. 1, pp. 45–66, 2015.

[24] R. Rojas-Cessa, Y. Kaymak, and Z. Dong, "Schemes for fast transmission of flows in data center networks," *Communications Surveys & Tutorials, IEEE*, vol. 17, no. 3, pp. 1391–1422, 2015.

[25] C. Wilson, H. Ballani, T. Karagiannis, and A. Rowtron, "Better never than late: Meeting deadlines in datacenter networks," in *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4. ACM, 2011, pp. 50–61.

[26] C.-Y. Hong, M. Caesar, and P. Godfrey, "Finishing flows quickly with preemptive scheduling," *ACM SIGCOMM Computer Communication Review*, vol. 42, no. 4, pp. 127–138, 2012.

[27] M. Alizadeh, S. Yang, M. Sharif, S. Katti, N. McKeown, B. Prabhakar, and S. Shenker, "pfabric: Minimal near-optimal datacenter transport," *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4, pp. 435–446, 2013.

[28] A. Munir, G. Baig, S. M. Irteza, I. A. Qazi, A. X. Liu, and F. R. Dogar, "Friends, not foes: synthesizing existing transport strategies for data center networks," in *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 4. ACM, 2014, pp. 491–502.

[29] H. Xu and B. Li, "Repflow: Minimizing flow completion times with replicated flows in data centers," *INFOCOM, 2014 Proceedings IEEE*, pp. 1581–1589, 2014.

[30] W. Bai, L. Chen, K. Chen, D. Han, C. Tian, and H. Wang, "Information-agnostic flow scheduling for commodity data centers," in *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, 2015, pp. 455–468.

[31] S. Boyd and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.

[32] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility," *Future Generation computer systems*, vol. 25, no. 6, pp. 599–616, 2009.

[33] A. Beloglazov, J. Abawajy, and R. Buyya, "Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing," *Future generation computer systems*, vol. 28, no. 5, pp. 755–768, 2012.

[34] T. Gunarathne, T.-L. Wu, J. Y. Choi, S.-H. Bae, and J. Qiu, "Cloud computing paradigms for pleasingly parallel biomedical applications," *Concurrency and Computation: Practice and Experience*, vol. 23, no. 17, pp. 2338–2354, 2011.

[35] T. Mathew, K. C. Sekaran, and J. Jose, "Study and analysis of various task scheduling algorithms in the cloud computing environment," in

Advances in Computing, Communications and Informatics (ICACCI, 2014 International Conference on. IEEE, 2014, pp. 658–664.

[36] Z.-H. Zhan, X.-F. Liu, Y.-J. Gong, J. Zhang, H. S.-H. Chung, and Y. Li, "Cloud computing resource scheduling and a survey of its evolutionary approaches," *ACM Computing Surveys (CSUR)*, vol. 47, no. 4, p. 63, 2015.

[37] L. Zuo, L. Shu, S. Dong, C. Zhu, and T. Hara, "A multi-objective optimization scheduling method based on the ant colony algorithm in cloud computing," *IEEE Access*, vol. 3, pp. 2687–2699, 2015.

[38] S. H. Adil, K. Raza, U. Ahmed, S. S. A. Ali, and M. Hashmani, "Cloud task scheduling using nature inspired meta-heuristic algorithm," in *2015 International Conference on Open Source Systems & Technologies (ICOSST).* IEEE, 2015, pp. 158–164.

[39] Z. Tang, L. Jiang, J. Zhou, K. Li, and K. Li, "A self-adaptive scheduling algorithm for reduce start time," *Future Generation Computer Systems*, vol. 43, pp. 51–60, 2015.

[40] C.-W. Tsai, W.-C. Huang, M.-H. Chiang, M.-C. Chiang, and C.-S. Yang, "A hyper-heuristic scheduling algorithm for cloud," *Cloud Computing, IEEE Transactions on*, vol. 2, no. 2, pp. 236–250, 2014.

[41] P. Kumar and A. Verma, "Independent task scheduling in cloud computing by improved genetic algorithm," *International Journal of Advanced Research in Computer Science and Software Engineering*, vol. 2, no. 5, 2012.

[42] G. Guo-ning, H. Ting-lei, and G. Shuai, "Genetic simulated annealing algorithm for task scheduling based on cloud computing environment," in *2010 International Conference on Intelligent Computing and Integrated Systems*, 2010.

[43] J. G. Koomey *et al.*, "Estimating total power consumption by servers in the us and the world," 2007.

[44] X. Zhu, L. T. Yang, H. Chen, J. Wang, S. Yin, and X. Liu, "Real-time tasks oriented energy-aware scheduling in virtualized clouds," *Cloud Computing, IEEE Transactions on*, vol. 2, no. 2, pp. 168–180, 2014.

[45] M. Liaqat, S. Ninoriya, J. Shuja, R. W. Ahmad, and A. Gani, "Virtual machine migration enabled cloud resource management: A challenging task," *arXiv preprint arXiv:1601.03854*, 2016.

[46] G. Han, W. Que, G. Jia, and L. Shu, "An efficient virtual machine consolidation scheme for multimedia cloud computing," *Sensors*, vol. 16, no. 2, p. 246, 2016.

[47] Y. Mhedheb, F. Jrad, J. Tao, J. Zhao, J. Kołodziej, and A. Streit, "Load and thermal-aware vm scheduling on the cloud," in *Algorithms and Architectures for Parallel Processing*. Springer, 2013, pp. 101–114.

[48] Y. Shen, Z. Bao, X. Qin, and J. Shen, "Adaptive task scheduling strategy in cloud: when energy consumption meets performance guarantee," *World Wide Web*, pp. 1–19, 2016.

[49] Q. Zhao, C. Xiong, C. Yu, C. Zhang, and X. Zhao, "A new energy-aware task scheduling method for data-intensive applications in the cloud," *Journal of Network and Computer Applications*, vol. 59, pp. 14–27, 2016.

[50] Z. Xiao, W. Song, and Q. Chen, "Dynamic resource allocation using virtual machines for cloud computing environment," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 24, no. 6, pp. 1107–1117, 2013.

[51] S. Shin, Y. Kim, and S. Lee, "Deadline-guaranteed scheduling algorithm with improved resource utilization for cloud computing," in *Consumer Communications and Networking Conference (CCNC), 2015 12th Annual IEEE*. IEEE, 2015, pp. 814–819.

[52] D. Liu and N. Han, "An energy-efficient task scheduler in virtualized cloud platforms," *International Journal of Grid and Distributed Computing*, vol. 7, no. 3, pp. 123–134, 2014.

[53] J. Li, M. Qiu, Z. Ming, G. Quan, X. Qin, and Z. Gu, "Online optimization for scheduling preemptable tasks on iaas cloud systems," *Journal of Parallel and Distributed Computing*, vol. 72, no. 5, pp. 666–677, 2012.

[54] Y. Ge and G. Wei, "Ga-based task scheduler for the cloud computing systems," in *Web Information Systems and Mining (WISM), 2010 International Conference on*, vol. 2. IEEE, 2010, pp. 181–186.

[55] A. Bohra and V. Chaudhary, "Vmeter: Power modelling for virtualized clouds," in *Parallel Distributed Processing, Workshops and Phd Forum*

(IPDPSW), 2010 IEEE International Symposium on, April 2010, pp. 1–8.

[56] A. Kansal, F. Zhao, J. Liu, N. Kothari, and A. A. Bhattacharya, "Virtual machine power metering and provisioning," in *Proceedings of the 1st ACM Symposium on Cloud Computing*, ser. SoCC '10, 2010, pp. 39–50.

[57] B. Krishnan, H. Amur, A. Gavrilovska, and K. Schwan, "Vm power metering: feasibility and challenges," *ACM SIGMETRICS Performance Evaluation Review*, vol. 38, no. 3, pp. 56–60, 2011.

[58] C. Gu, H. Huang, and X. Jia, "Power metering for virtual machine in cloud computing-challenges and opportunities," *Access, IEEE*, vol. 2, pp. 1106–1116, 2014.

[59] N. Kim, J. Cho, and E. Seo, "Energy-credit scheduler: an energy-aware virtual machine scheduler for cloud systems," *Future Generation Computer Systems*, vol. 32, pp. 128–137, 2014.

[60] B. Veeravalli, D. Ghose, V. Mani, and T. G. Robertazzi, "Scheduling divisible loads in parallel and distributed systems," *Los Almitos: IEEE Computer Society Press, California*, 1996.

[61] T. G. Robertazzi, "Ten reasons to use divisible load theory," *Computer*, vol. 36, no. 5, pp. 63–68, 2003.

[62] Z. Zhang and T. G. Robertazzi, "Scheduling divisible loads in gaussian, mesh and torus network of processors," *IEEE Transactions on Computers*.

[63] "Dynamic scaling of cpu and ram for vms in apache cloudstack," https://cwiki.apache.org/confluence/display/CLOUDSTACK/Dynamic+scaling+of+CPU+and+RAM.

[64] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 4. ACM, 2008, pp. 63–74.

[65] J. L. Hellerstein, W. Cirne, and J. Wilkes, "Google cluster data," *Google research blog, Jan*, 2010.

[66] L. M. Leemis, *Reliability: probabilistic models and statistical methods*. Prentice-Hall, Inc., 1995.

[67] J. Lin and D. Ryaboy, "Scaling big data mining infrastructure: the twitter experience," *ACM SIGKDD Explorations Newsletter*, vol. 14, no. 2, pp. 6–19, 2013.

[68] A. Singh, J. Ong, A. Agarwal, G. Anderson, A. Armistead, R. Bannon, S. Boving, G. Desai, B. Felderman, P. Germano *et al.*, "Jupiter rising: A decade of clos topologies and centralized control in google's datacenter network," in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*. ACM, 2015, pp. 183–197.

[69] D. Borthakur, "Hdfs architecture guide," *HADOOP APACHE PROJECT http://hadoop. apache. org/common/docs/current/hdfs design. pdf*, 2008.

[70] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The google file system," in *ACM SIGOPS operating systems review*, vol. 37, no. 5. ACM, 2003, pp. 29–43.

[71] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[72] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly, "Dryad: distributed data-parallel programs from sequential building blocks," in *ACM SIGOPS Operating Systems Review*, vol. 41, no. 3. ACM, 2007, pp. 59–72.

[73] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets." *HotCloud*, vol. 10, pp. 10–10, 2010.

[74] M. Chowdhury and I. Stoica, "Coflow: A networking abstraction for cluster applications," in *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*. ACM, 2012, pp. 31–36.

[75] M. Chowdhury, M. Zaharia, J. Ma, M. I. Jordan, and I. Stoica, "Managing data transfers in computer clusters with orchestra," *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4, pp. 98–109, 2011.

[76] F. R. Dogar, T. Karagiannis, H. Ballani, and A. Rowstron, "Decentralized task-aware scheduling for data center networks," in *ACM SIG-*

COMM Computer Communication Review, vol. 44, no. 4. ACM, 2014, pp. 431–442.

[77] M. Chowdhury, Y. Zhong, and I. Stoica, "Efficient coflow scheduling with varys," in *Proceedings of the 2014 ACM conference on SIGCOMM*. ACM, 2014, pp. 443–454.

[78] M. Chowdhury and I. Stoica, "Efficient coflow scheduling without prior knowledge," in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*. ACM, 2015, pp. 393–406.

[79] Y. Zhao, K. Chen, W. Bai, 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.

[80] Y. Peng, K. Chen, G. Wang, W. Bai, Z. Ma, and L. Gu, "Hadoopwatch: A first step towards comprehensive traffic forecasting in cloud computing," in *INFOCOM, 2014 Proceedings IEEE*. IEEE, 2014, pp. 19–27.

[81] A. P. Punnen, "A linear time algorithm for the maximum capacity path problem," *European Journal of Operational Research*, vol. 53, no. 3, pp. 402–404, 1991.

[82] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Foundations and Trends® in Machine Learning*, vol. 3, no. 1, pp. 1–122, 2011.

[83] I. Necoara and D. Clipici, "Efficient parallel coordinate descent algorithm for convex optimization problems with separable constraints: application to distributed mpc," *Journal of Process Control*, vol. 23, no. 3, pp. 243–253, 2013.

[84] V. Cevher, S. Becker, and M. Schmidt, "Convex optimization for big data: Scalable, randomized, and parallel algorithms for big data analytics," *Signal Processing Magazine, IEEE*, vol. 31, no. 5, pp. 32–43, 2014.

[85] Y. Guo, J. Rao, and X. Zhou, "ishuffle: Improving hadoop performance with shuffle-on-write." in *ICAC*. Citeseer, 2013, pp. 107–117.

[86] M. Li, L. Zeng, S. Meng, J. Tan, L. Zhang, A. R. Butt, and N. Fuller, "Mronline: Mapreduce online performance tuning," in *Proceedings of the 23rd international symposium on High-performance parallel and distributed computing*. ACM, 2014, pp. 165–176.