TIME DIVISION DOMINANT RESOURCE ALLOCATION IN CLOUD ENVIRONMENTS

by

Xu Zhang

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

in

Communications

Department of Electrical and Computer Engineering
University of Alberta

© Xu Zhang, 2015

Abstract

Providing service in cloud is a popular trend nowadays. Users request resources from a shared pool in a framework like Hadoop, and the Hadoop fair scheduler controls the progress of tasks from users.

In cloud, usually multiple resource types are available. The heterogeneous user demands and competition between users in cloud make the fairness control of multiple-resource allocation much more complicated than the single-resource allocation. A recent work proposed a Dominant Resource Fairness (DRF) method for multiple-resource allocation, which has been included in Hadoop Next-Generation.

However, DRF may not lead to desired fairness performance as a result of the indivisibility of the user demands. This thesis proposes a Time Division Allocation (TDA) method with time slots for two users. The TDA method allocates resources over the time slots and each time slot is assigned a different allocation. By adjusting the length of the time slots and the resource allocations, TDA method can achieve a global optimal max-min fairness. A further study shows that two time slots are sufficient to achieve optimality.

Besides the analysis about fairness, this thesis also explores the theoretical performance bounds for two users. This analysis illustrates the relationship between

user demands and the maximum dominant resource share. The evaluation shows that the TDA method performs better than the DRF method.

Table of Contents

List of Tables

List of Figures

List of Symbols

$\mathbf{a}_i$	the ith allocation in allocation set
a_{iu}	the number of tasks assigned to user u in allocation i;
$ADRS_u$	average dominant resource share of user u
$\mathbf{b}_j$	resource allocation vector
b_{iu}	dominant resource share in allocation i for user u
$\mathbf{c}$	total capacity of the resources
$\mathbf{d}_u$	demand vector of user u
d_{uk}	the demand of user u for resource k
$\mathcal{G}$	allocation groups
K	number of time slots
m	number of resource types
n	number of the users in the system
$q_{r,u}$	normalized resource share ratio compared to the dominant resource.
Q	the popular factor for the most demanded resource
$\mathcal{R}$	set of all the possible resource allocation
r_{iu}	dominant resource share for user u in allocation i
s_u	normalized dominant resource share for user u
T	transpose operation
t_i	time duration for the ith time slot

List of Abbreviations

Acronyms	Definition
ADRS	Average Dominant Resource Share
DRF	Dominant Resource Fairness
DRS	Dominant Resource Share
FIFO	First In First Out
HDFS	Hadoop Distributed File System
IaaS	Infrastructure as a service
PaaS	Platform as a service
SaaS	Software as a service

Chapter 1

Introduction

1.1 Cloud Computing

Cloud computing is an Internet platform widely deployed with a big impact in modern life. The transformation from traditional service paradigm to cloud computing environments is a popular trend [1]. A well-recognized definition of cloud computing from the National Institute of Standards and Technology -US Department of Commerce(NIST) [2] is that: "Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction."

The cloud services can be offered in three famous models: Infrastructure as a service (IaaS) where some basic services are provided including virtual machines, storage and network resources (e.g., Amazon S3), Platform as a service (PaaS) which offers environments like database, programming compiling environment or even operation system (e.g., MicroSoft Azure), or Software as a service (SaaS) when some software services are provided (e.g., Google Apps) [2] [3]. IaaS is currently the most basic but successful cloud service model extensively supported by many cloud service providers such as Amazon EC2 [4]. IaaS users utilize services in this environment including storage of web data, and request computing capability

from a cluster of data centers.

Many cluster computing frameworks, such as Hadoop, Dryad and Mesos, provide services for users in an IaaS model [5][6][7][8]. Hadoop, a widely used distributed cloud framework by companies like Amazon and Facebook, allows for the distributed processing of large data sets across clusters of computers [5]. Hadoop's architecture consists of two major components: the storage component called the Hadoop Distributed File System (HDFS) is used to store files in replicas across multiple machines [9], and the processing component consisting of "map" function and "reduce" function, called MapReduce, is to process input files and generate large output datasets in parallel [10][11].

The MapReduce process is shown in Figure 1.1. The MapReduce engine has a master (called JobTracker) that is responsible for the MapReduce jobs submitted by the users. The map function, which is defined by the user, takes input data files and produces intermediate results (map task). After that, the reduce function, which is also defined by the user, accepts the intermediate results and finishes the final processing (reduce task) [11]. The master controls a number of machines in the cluster as workers (called TaskTrackers) and assigns the tasks to available workers. The workers can communicate with the master through a frequent heartbeat (message of status).

The map function and reduce function are both defined by users, and the resource demands vary between tasks. The map task usually requires more CPU cores while the reduce task usually consumes more memory. Thus, in current design, two types of resource slots are defined, map slot and reduce slot [12], each with a pre-defined amount of resources. Each worker will be assigned a number of map slots and a number of reduce slots by the master. Although this resource allocation method is not actually based on demands of the tasks, the method is frequently deployed because it is easy to manage [2] [13].

Since the resources in cloud computing are shared by users, it is important to achieve fairness among users. Fairness does not only mean simply equalizing the share of the users. It also means controlling or scheduling for a global optimization

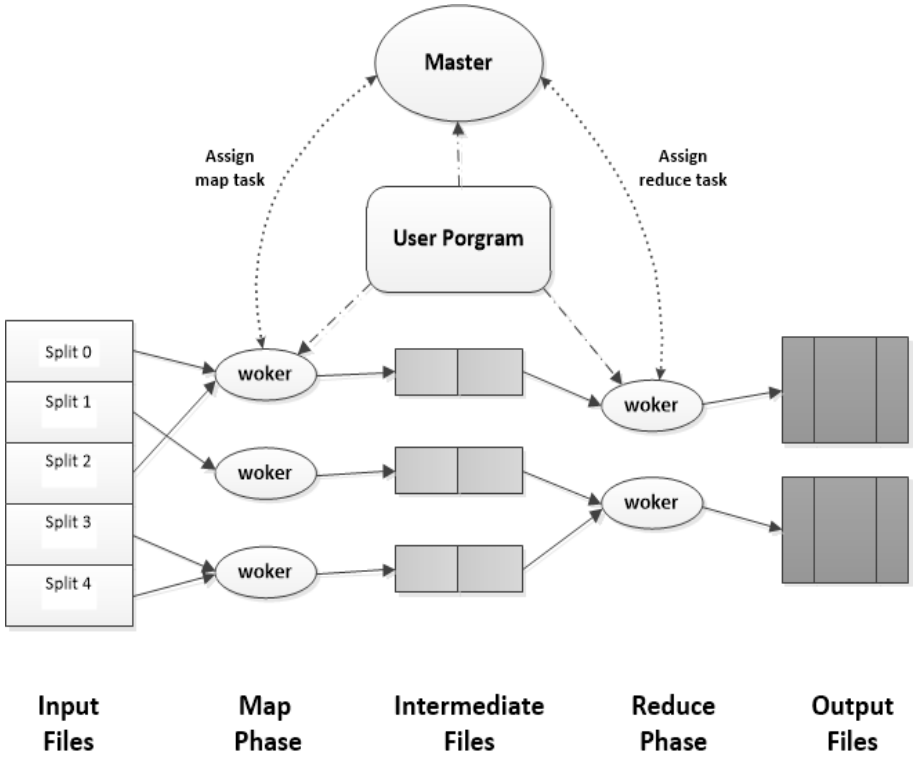

Fig. 1.1. MapReduce Process

for the system. This is why fairness is always a critical consideration. By controlling the resource allocation, the usage of users can be regulated and the progress of different tasks can be balanced.

Fairness has been well investigated in the literature, for example, the fairness of channel sharing in communication systems. Compared to the fair resource allocation in communication systems, the resource allocation in a cloud environment is much more challenging due to the following reasons [14].

- In a communication system, there is only one resource (e.g., spectrum bandwidth). In a cloud environment, in order to satisfy the various demands from cloud service users, two or more types of resources are needed. For example, MapReduce jobs consume both CPU cores and memory.

- The demands of tasks submitted from users are highly heterogeneous in terms of the amount of different types of resources needed. For example, some tasks may need a lot of CPU cores, while others may need a lot of memory

spaces. The multidimensional resource allocation in a cloud environment is much more complicated than the single-dimension resource allocation in a communication system.

- Users in a cloud environment may lie about their demands to gain extra benefits (usually means more resource share).

The default scheduling model in Hadoop is a First-In-First-Out (FIFO) scheduler [13]. This scheduler is easy to implement but some large jobs with long processing time will delay the whole operation. Besides FIFO, in some scenarios the system has a fair sharing scheduler with job priorities [13]. In this scheduler, the allocation of resource slots is optimized on time average.

The current scheduling strategy of resource allocation is neither fair nor efficient. There is no fairness control in a simple FIFO model, and the fair sharing is optimized from time prospective not from resources prospective. A research in [15] has analyzed the usage of a data center cluster at Twitter for over one month. The result shows that: for CPU, although 80 % total capacity are reserved, the cumulative CPU usage is only about 20%. For memory, the usage rate is under 50% for the same reservation rate of 80%. There is also a study [16] indicating that the CPU utilization at Amazon EC2 is below 10%.

The resource allocation and scheduling strategy in current model of cloud computing are quite naive, which limits the overall performance of the cloud service. First, the resources allocated for tasks are in a granularity of a resource slot. For example, if a resource slot includes 10 CPU cores and 10GB memory, and a task needs 1 CPU core and 1GB memory, the task still needs to be allocated one resource slot, resulting in resource waste. Second, during the allocation process, fairness is not well controlled or sometimes even not considered.

There are numerous research efforts trying to elevate the performance including utility optimization, task priority scheduling, prediction, and fairness among users [2] [17].

Some works are limited in the direction of single resource [18]. The work in

[19] proposes a scheduler considering fairness for distributed clusters. But this scheduler has difficulty in supporting multiple resources. Other works such as [12] discuss multiple resource allocation in resource slots. All these allocation models fail to adapt to the demand heterogeneity and resource competition situation in cloud environments.

A recent work [14] makes a big progress in this area. To balance tasks with different resource demands, this work comes up with an original definition of "fairness" for multiple resources. In this work, the fairness is measured by each user's maximum normalized resource share, called dominant resource share. Based on this fairness measurement, a multiple-resource allocation strategy is proposed. The resources are no longer allocated in units of resource slots, but can be controlled based on the real demands of the tasks. Details of the multiple-resource allocation strategy are given in Chapter 2. This *dominant resource allocation* method is implemented in Hadoop Next-Generation (YARN) for resource management and scheduling [20][21]. The Mesos platform employing *dominant resource allocation* method proposed in paper [7] is used at Twitter and other organizations [8].

1.2 Thesis Motivations

Dominant Resource Fairness in [14] gives a novel solution to the multiple-resource allocation in cloud computing environments. The authors of [14] present features of heterogeneous resource demands in data centers. According to the analysis of the demands, the drawback of resource slot based allocation is identified. To adapt to the computing environment with competition and heterogeneous users, the authors come up with a practical Dominant Resource Fairness (DRF) Allocation method.

Following works extend DRF in various ways. Works in [22] and [23] propose a unify framework to describe the fairness and efficiency trade-offs. The work in [18] analyzes the strength and limitation of DRF. The work in [24] generalizes this allocation method to a more complicated scenario with heterogeneous cloud servers. The work in [7] implements a framework using DRF method. Other works

like [25] [26] extend DRF in network resources and packets processing.

Compared to the resource slot based allocation (in which a task is assigned a number of resource slots), the DRF allocation based on task demand seems to be more delicate in resource management. In this thesis, using the same dominant resource definition, we propose to achieve better fairness and efficiency in a time sharing mode. In specific, we propose that the time in the system is divided (usually not equally divided) to a number of intervals, and in each interval, a resource allocation scheme is implemented. We target at the overall fairness and resource utilization efficiency, by determining the number of intervals needed and the resource allocations in the intervals.

1.3 Thesis Outline

This thesis consists of six chapters. Chapter 1 is the introduction, in which a overview of the cloud service configuration and the fairness and efficiency issues in cloud computing are presented. In Chapter 2, we have a detailed illustration of the multiple-resource allocation background. In Chapter 3, we propose the Time Division Allocation (TDA) method which improves the fairness and efficiency in cloud computing. In Chapter 4, we discuss the theoretical upper bounds that the time division allocation can achieve. Chapter 5 presents performance evaluation of our allocation method. Chapter 6 includes the conclusion and future work.

Chapter 2

Background and Literature Review

In this chapter, we will have a review of the background in cloud resource allocation. The challenge in allocating multiple resources and the drawback of single-resource allocation method are identified. Then a review of the procedure of DRF allocation is given.

2.1 Heterogeneous Tasks of Users

One major challenge in a multiple-resource environment is the heterogeneity in user tasks [27]. Figure 2.1-2.4 illustrate the tasks' resource usage statistics from Google. Demands of 3,535,029 tasks belong to 4 job types submitted to the clusters ranging widely in both CPU and memory. These figures show a normalized CPU core and memory demand. The original usage statics are obscured by Google using a linear transform [28][29][30] [31], but the characterization of resource demands is reserved. Since CPU is allocated by cores, the CPU demands take discrete values. The demand of memory is in a continuous manner. Another fact of the task demands is the distinction among different types of jobs. As shown in Figure 2.1-2.4, there are 4 different types of jobs in these 3,535,029 workloads. Tasks in job type 0 mainly demand memory, while Type 1 requires more CPU cores. Type 2 may consist of some large tasks because the demands of CPU and memory are relatively large, while the tasks of job type 3 can be considered as memory-heavy tasks.

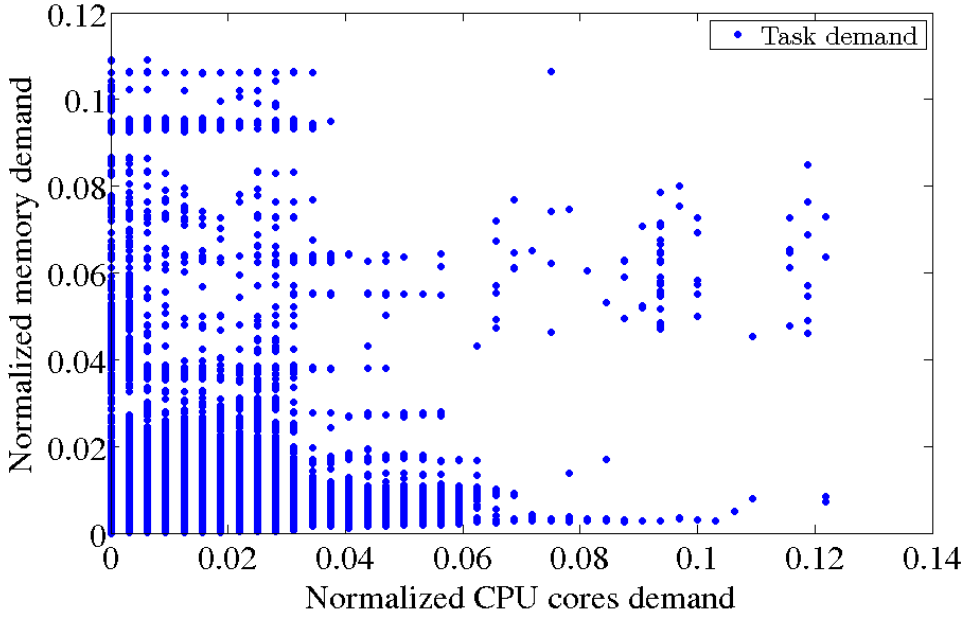

Fig. 2.1. Heteregerous Task Demand of Job Type 0 from Goolge Cluster Data

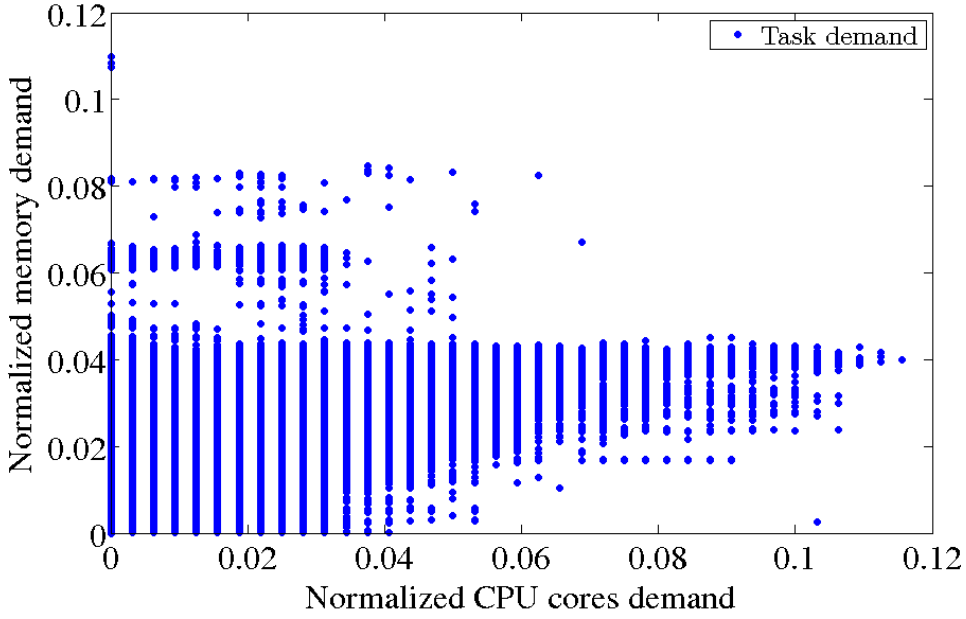

Fig. 2.2. Heteregerous Task Demand of Job Type 1 from Goolge Cluster Data

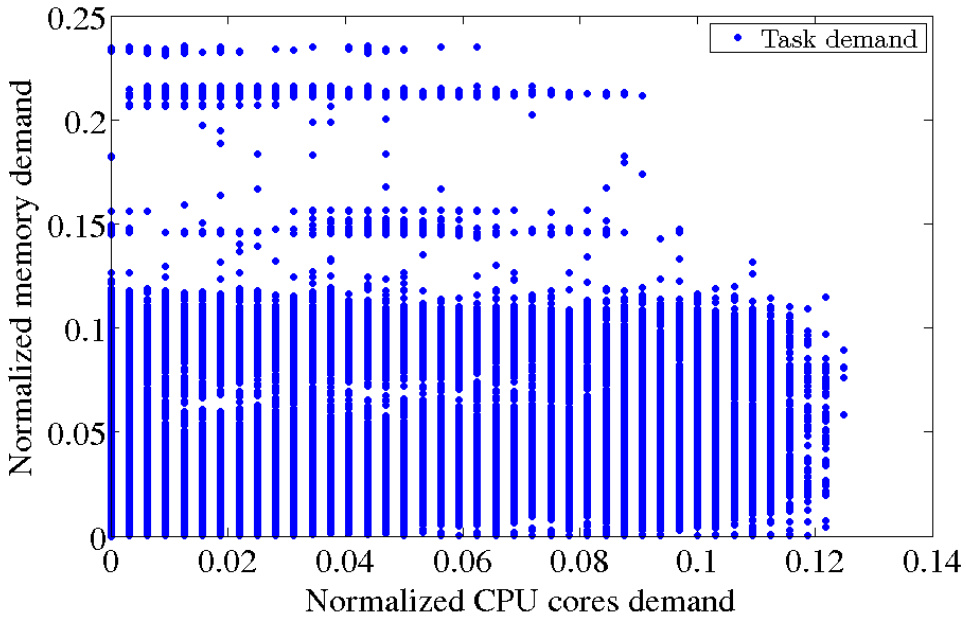

Fig. 2.3. Heteregerous Task Demand of Job Type 2 from Goolge Cluster Data

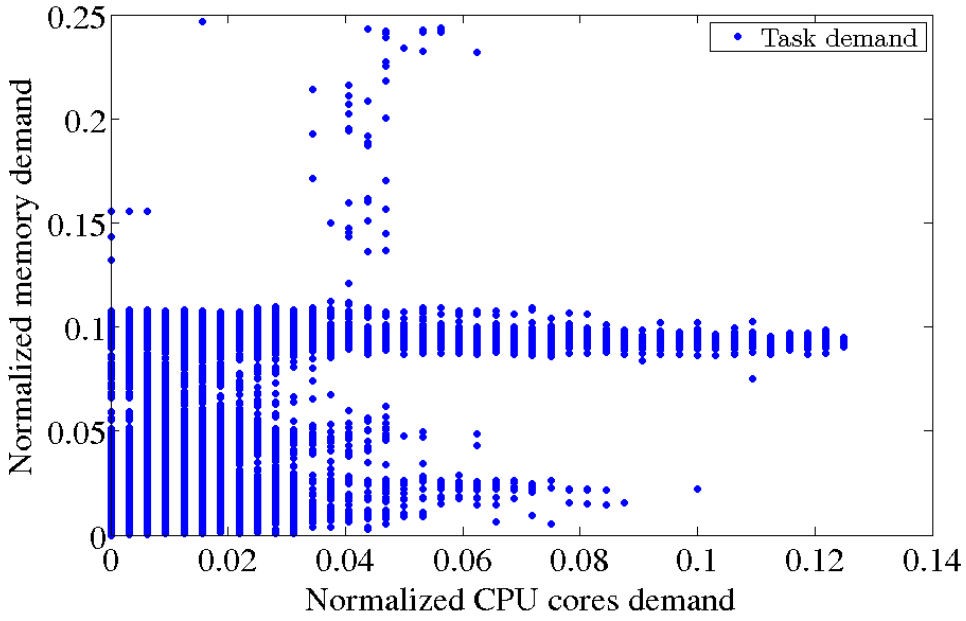

Fig. 2.4. Heteregerous Task Demand of Job Type 3 from Goolge Cluster Data

Based on this investigation, it is not difficult to understand why allocations using resource slots may lead to poor performance in fairness and result in an under-utilization of the system. The resource slot based allocation only has two different types of resource slot, each with a pre-defined amount of CPU cores and memory space. But the actual demand distribution is far more complicated. Therefore, the resource slot allocation is unsuitable for resource allocation for various demands in an environment with multiple resources.

2.2 Single-Resource Allocation Methods

As a key concept in cloud computing, resource allocation is discussed in many papers. Even though the situation for multiple resources is different from that with a single-resource, one straightforward idea in cloud computing is to use single-resource allocation.

For single-resource allocation, max-min fairness is a widely used criterion [32]. The major point of max-min fairness is trying to maximize the share of the user with minimum resources. For example, the max-min allocation of CPU in [33]. As another example, the work in [34] [35][36] use max-min fairness for link bandwidth.

Despite that max-min fairness successfully solved lots of problems in single-resource allocation, the extension to multiple-resources allocation is difficult. In single-resource allocation, the resource share of a user can be described as the ratio of the amount of allocated resources to the amount of total resources. However, in multiple resource scenarios, for a user with different demands in multiple-resource environment, the ratio of the amount of allocated resources to the amount of total resources are different for one resource type to another. Therefore, it is challenging to select a suitable fairness criterion.

In the popular computing framework Hadoop, this multiple-resource problem is transformed to a single-resource problem by using resource slots [13]. For each resource slot, it contains a fixed amount of resources. For example, in the MapRe-

duce process[11], there are two types of slots, one with more CPU, mainly for map tasks, and the other with more memory, mainly for reduce tasks. Apparently the heterogeneity of the user tasks is not considered. As a result, the performance (the utilization of the resources) is not ideal. It is shown that almost 70% of the resources reserved in Amazon EC2 are wasted [15] [16].

Some research efforts like [19] try to achieve a better performance through a fairness scheduling based on the running time of different tasks. But it still does not work well with multiple resources. Other works [12] also attempt to solve the resource allocation problem with different schedulers, but continue to use resource slots as the allocation unit. The Fair scheduler [37] has a two-level architecture. In the first level Hadoop allocates resources to different resource pools based on a max-min fairness on resource slots. In the second level, each resource pool allocates resources to users. The whole process is similar to the process of pouring water in different buckets.

2.3 Multiple-Resource Allocation

The resource slot based allocation tries to simplify the problem to a single-resource allocation problem, but it has been proven to be inefficient. Therefore, a completely different resource allocation method is needed, in which a new definition of resource share in a multiple-resource environment is necessary.

For multiple-resource allocation in cloud environments, some specific properties should be satisfied [14] [38].

- **Sharing Incentive:** The multiple-resource allocation for any user should be better than an even partition of all the resources. Since the default allocation is by using resource slots, the multiple resource allocation should be at least not worse than the resource slot based allocation.

- **Strategy-proof:** No user can benefit through a faked demand in competing resources with other users. This is a requirement to avoid unfair competition

between users.

- **Pareto Efficiency:** No one can obtain higher resource share without reducing the share of another user. This property results in an optimization of the system utilization.

According to these requirements, some allocation ideas turn out to be failures. Suppose a situation when the system contains two types of resources: CPU and memory. The capacity is represented as $< 14\ CPU, 14\ GB >$. The demand of a task from user 1 and user 2 are $< 1\ CPU, 1\ GB >$, $< 2\ CPU, 4\ GB >$.

One straightforward resource allocation is to imaginarily assume the resource demand values for different resource types have the same unit. Thus, for user 1 the resource demand is 2 per task while for user 2 the demand is 6 per task. From fairness perspective, the allocation for two users are $< 6\ CPU, 6\ GB >$, $< 4\ CPU, 8\ GB >$, respectively. However, for user 1 the allocation is worse than an even partition $< 7\ CPU, 7\ GB >$, which is not good.

2.4 Dominant Resource Fairness

To deal with multiple-resource allocation, a new measurement is needed to embody the heterogeneity of tasks and reflect the usage share comparison between users. Recently, the work in [14] proposed the notion of *dominant resource* to solve this problem. For a resource type, normalized resource share of a user is defined as the ratio of the user's demand of this resource to the total amount of this resource. Dominant resource for a certain user is the resource that has the maximum normalized resource share among all the resource types.

For example, suppose the system capacity is $< 16\ CPU, 12\ GB >$, the demand of a task from two users are $< 6\ CPU, 1.5\ GB >$, $< 1\ CPU, 3\ GB >$. According to the definition, the normalized resource shares for user 1 and user 2 are $< \frac{3}{8}, \frac{1}{8} >$ and $< \frac{1}{16}, \frac{1}{4} >$. User 1's dominant resource is CPU, while user 2's dominant resource is memory.

Round	Round 1	Round 2	Round 3	Round 4	Round 5
user 1	<**6, 1.5**>	$< 6, 1.5 >$	$< 6, 1.5 >$	<**12, 3**>	$< 12, 3 >$
user 2	$< 0, 0 >$	<**1, 3**>	<**2, 6**>	$< 2, 6 >$	<**3, 9**>
DRS compare	$\frac{3}{8} > 0$	$\frac{3}{8} > \frac{1}{4}$	$\frac{3}{8} < \frac{1}{2}$	$\frac{3}{4} > \frac{1}{2}$	$\frac{3}{4} = \frac{3}{4}$
Remaining	$< 10, 10.5 >$	$< 9, 7.5 >$	$< 8, 4.5 >$	$< 2, 3 >$	$< 1, 0 >$

TABLE 2.1
DOMINANT RESOURCE FAIRNESS ALLOCATION PROCESS.

The allocation scheduling process is a max-min fairness allocation based on dominant resource share, referred to as *dominant resource fairness (DRF) allocation*. In each round, the allocation algorithm allocates resources to the user that has obtained minimum dominant resource share in previous rounds.

The allocation process is shown in Table 2.1, in which bold font means the allocation in the current round, and "DRS compare" means comparison between dominant resource shares of the two users. In Round 1, a user is randomly picked, which is user 1. A task from user 1 is allocated, which consumes $< 6\ CPU, 1.5\ GB >$. So the dominant resource share of user 1 is $\frac{6}{16} = \frac{3}{8}$, while dominant resource share of user 2 is 0. Since user 2 has a less amount of dominant resource share in round 1, a task from user 2 is allocated in round 2, which consumes $< 1\ CPU, 3\ GB >$. So after round 2, user 2's dominant resource share is $\frac{3}{12} = \frac{1}{4}$, while user 1's dominant resource share is still $\frac{3}{8}$. Since user 2 still has a less amount of dominant resource share after round 2, a task from user 2 is allocated in round 3, and thus, the total resources allocated for user 2 are $< 2\ CPU, 6\ GB >$. After round 3, user 2's dominant resource share is $\frac{6}{12} = \frac{1}{2}$, while user 1's dominant resource share is still $\frac{3}{8}$. Since user 1 has a less amount of dominant resource share after round 3, a task from user 1 is allocated in round 4. This procedure continues until after round 5, when the remaining resource amount $< 1\ CPU, 0\ GB >$ is not sufficient for a task from any user. Figure 2.5 shows the resource allocation result.

The idea of dominant resource fairness satisfies the properties introduced in Section 2.3.

- It can be proven that the DRF allocation is sharing incentive. The above ex-

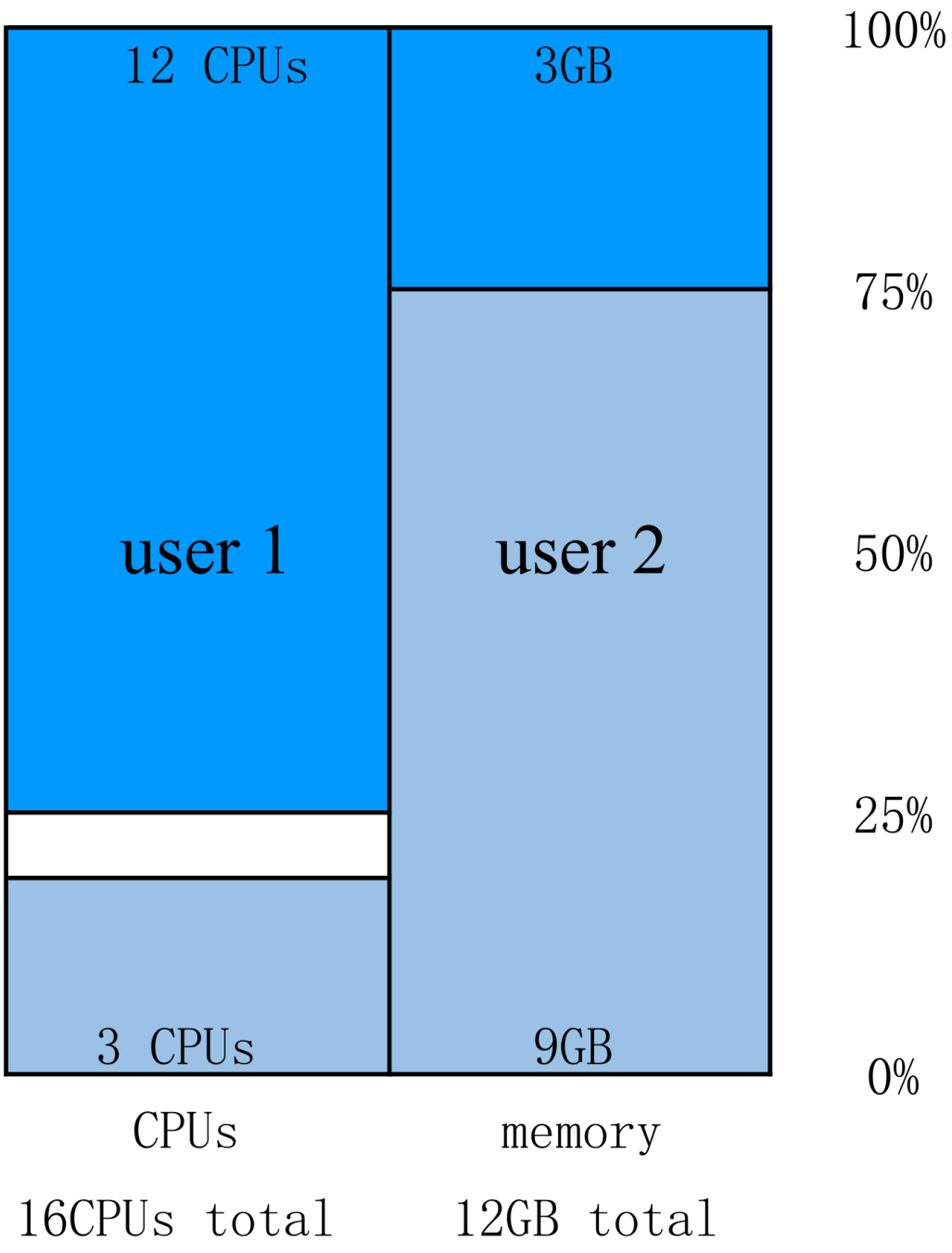

Fig. 2.5. Dominant Resource Fairness Allocation Result

ample shows that the allocation is performed according to the demands of all the users, and each user receives a better allocation than even partition resource allocation (i.e., each user is allocated 8 CPU cores and 6 GB memory).

- No users can benefit from lying about the demand. Since the allocation is based on dominant resource, there is no benefit to lie about the non-dominant resources. The allocation is based on the "actual" share of dominant resource. So lying about the dominant resource demand does not bring about benefit either.

- At last, the allocation stops until there are no enough resources for a task from any user. Thus, no one can increase the resource share without sacrificing another user, which proves the Pareto Efficiency property.

DRF allocation has attracted a lot of interests in research because of the novel idea in multiple-resource allocation in cloud environments. Work in [39] discusses DRF method and proposed another max-min fairness idea. Based on the requirements mentioned in [14], the work in [39] proposes an extension of max-min fairness method which provided support for constraints. The constraints include conditions, e.g., that tasks should work in the same virtual machine, in a specific operating environment, using special hardware and so on. An algorithm is also provided in the paper that implements this mechanism. The DRF method is extended to packet processing area in [40]. This paper focuses on the problem about the fairness of resource allocation in controlling network-flows. When network-flows go through the "middleboxes", multiple resources like CPU, bandwidth are required by the "middleboxes" to process the flows. Queueing for different flows requires a fairness control for multiple resources. Thus the DRF-Queueing method in this paper provides a solution for this problem. Work in [41] proposes an Hierarchical-DRF based on DRF method, which is designed to support organizational hierarchy. This work is for the scenario that when the resources are used between different departments in one organization, multiple resources are shared in a tree architecture. In industry, DRF is included in Hadoop Next-Generation (YARN) [20][21].

Mesos, a resource sharing layer framework using DRF, is widely used by company like Twitter [7][8].

Chapter 3

Optimal Time Division Resource Allocation

In this chapter, we focus on the time division idea for resource allocation. First, we use a simple example to demonstrate the benefit of the time division idea. After that, we formulate the time division dominant resource allocation problem and find its optimal solution.

3.1 A Simple Example

Recall that the major objective of DRF allocation is to maximize the minimum dominant resource share of the users.

Consider a system with two types of resources, and the capacity of the two resources is given as $< 15, 15 >$. There are two users, user 1 and user 2. The resource demand of a task from user 1 is $< 5, 2 >$. The resource demand of a task from user 2 is $< 3, 3.5 >$. So user 1's dominant resource is the first resource type, while user 2's dominant resource is the second resource type. Based on DRF allocation, user 1 gets resources $< 5, 2 >$, and user 2 gets resources $< 9, 10.5 >$. Accordingly, in the DRF allocation, the dominant resource share of the two users are $\frac{5}{15}$ and $\frac{10.5}{15}$.

Now consider two time slots with equal length. In the first time slot, user 1 gets

resources $< 15, 6 >$, while user 2 gets nothing. In the second time slot, user 1 gets nothing, while user 2 gets $< 12, 14 >$. So over the two slots, the dominant resource share of user 1 is $\frac{15}{15}$ and $\frac{0}{15}$, respectively, with an average being $\frac{7.5}{15}$. For user 2, its dominant resource share over the two time slots is $\frac{0}{15}$ and $\frac{14}{15}$, respectively, with an average being $\frac{7}{15}$.

It is apparent that the minimum dominant resource share of the users in the two-slot case ($\frac{7}{15}$) is higher than that in the DRF allocation ($\frac{5}{15}$).

This example demonstrates that by using more time slots, we may achieve better fairness in terms of maximizing the minimum dominant resource share of the users.

In the above example, only two time slots with equal length are considered, and in each time slot, an extreme resource allocation (i.e., one user is allocated as many resources as possible, while the other user is allocated nothing). In general, for overall optimization, the following questions should be answered: how many slots should be used? How long is each slot? and what is the resource allocation in each slot?

3.2 Problem Formulation

Consider a system with two users and a number, m, of resource types. The total capacity of the resources are given as $\mathbf{c} = [c_1, c_2, ..., c_m]^T$, where superscript T means transpose operation. For user 1, the resource demand of one task is given as $[d_{11}, d_{12}, ..., d_{1m}]^T$, in which d_{1i} ($i \in \{1, 2, ..., m\}$) is the demand of resource type i by a task of user 1. For user 2, the the resource demand of one task is given as $[d_{21}, d_{22}, ..., d_{2m}]^T$. For each user, based on the total available capacity of the m resources and the resource demands of a task of the user, the dominant resource of the user can be identified, and the normalized resource share of the dominant resource can be calculated, denoted as $s_1 = \max_{i\in\{1,2,...,m\}} \frac{d_{1i}}{c_i}$ for user 1 and $s_2 = \max_{i\in\{1,2,...,m\}} \frac{d_{2i}}{c_i}$ for user 2.

Consider the total time duration is 1. The total time duration is divided into a number, K, of time slots, with duration $t_1, t_2, ..., t_K$, respectively. So we have

$t_1 + t_2 + ... + t_K = 1$.

In each time slot, a resource allocation is implemented. For resource utilization efficiency, here we consider only the resource allocations that satisfy a Saturation Property as follows.

Saturation Property: In each resource allocation, the remaining (unallocated) resources are not sufficient for a task from any user.

All the possible allocations can be listed using an enumeration method. Assume that we have totally N such resource allocations (satisfying the Saturation Property), denoted as $\mathbf{a}_1, \mathbf{a}_2, ..., \mathbf{a}_N$. Here $\mathbf{a}_i$ ($i \in \{1, 2, ..., N\}$) can be represented as $\mathbf{a}_i = [a_{i1}, a_{i2}]^T$, with a_{i1} and a_{i2} being the number of accommodated tasks from user 1 and user 2, respectively, in resource allocation $\mathbf{a}_i$. Denote the set of all possible resource allocations as $\mathcal{A} \triangleq \{\mathbf{a}_1, \mathbf{a}_2, ..., \mathbf{a}_N\}$.

In resource allocation $\mathbf{a}_i$, the dominant resource share of user 1 and user 2 are given as $b_{i1} \triangleq a_{i1}s_1$ and $b_{i2} \triangleq a_{i2}s_2$, respectively. For presentation simplicity, we also call vector $\mathbf{b}_j \triangleq [b_{j1}, b_{j2}]^T$ as a resource allocation. This definition is used in the sequel of this thesis. And the set of all possible resource allocations is denoted as $\mathcal{R} \triangleq \{\mathbf{b}_1, \mathbf{b}_2, ..., \mathbf{b}_N\}$.

Over the K time slots, K resource allocations are implemented, denoted as $\mathbf{r}_1, \mathbf{r}_2, ..., \mathbf{r}_K \in \mathcal{R}$. Without loss of generality, the K resource allocations are assumed to be different from each other.[1] Then the *average dominant resource share* of user 1 and user 2 over the K slots are given as $\sum_{j=1}^{K} r_{j1}t_j$ and $\sum_{j=1}^{K} r_{j2}t_j$, respectively, where r_{j1} and r_{j2} are the first and second element in resource allocation $\mathbf{r}_j$, respectively.

The objective is to maximize the minimum average dominant resource share of the two users. Thus, we formulate the following time-resource allocation problem:

$$\begin{aligned} \max_{K, \mathbf{r}_1, \mathbf{r}_2, ..., \mathbf{r}_K \in \mathcal{R}, t_1, t_2, ..., t_K} \quad & \min\left(\sum_{j=1}^{K} r_{j1}t_j, \sum_{j=1}^{K} r_{j2}t_j\right) \\ \text{subject to:} \quad & t_1 + t_2 + ... + t_K = 1. \end{aligned} \tag{3.1}$$

[1] If two resource allocations over two slots are the same, then we can combine the two slots into one single slot.

In this formulation, constraints $0 \leq t_1 \leq 1$, $0 \leq t_2 \leq 1$,...,$0 \leq t_K \leq 1$ are by default and are omitted for brevity. Also, constraint that $\mathbf{r}_1, \mathbf{r}_2, ..., \mathbf{r}_K$ are different from each other is by default for the problem formulation, and is omitted here for brevity.

All resource allocations in $\mathcal{R}$ can be grouped into three sets: set $\mathcal{G}_1$ includes all resource allocations in which the dominant resource share of user 1 is higher than that of user 2; set $\mathcal{G}_2$ includes all resource allocations in which the dominant resource share of user 2 is higher than that of user 1; and set $\mathcal{G}_3$ includes all resource allocations in which the dominant resource shares of user 1 and user 2 are equal. Therefore, we have $\mathcal{R} = \mathcal{G}_1 \cup \mathcal{G}_2 \cup \mathcal{G}_3$.

To solve Problem (3.1), we have three cases:

- Case I: $\mathcal{G}_1 = \Phi$ (null set) or $\mathcal{G}_2 = \Phi$. In other words, in any resource allocation in $\mathcal{R}$, the dominant resource share of a user is always higher than or equal to that of the other user;
- Case II: $\mathcal{G}_1 \neq \Phi$, $\mathcal{G}_2 \neq \Phi$, and $\mathcal{G}_3 = \Phi$;
- Case III: $\mathcal{G}_1 \neq \Phi$, $\mathcal{G}_2 \neq \Phi$, and $\mathcal{G}_3 \neq \Phi$.

Before we solve Problem (3.1) in the three cases, we have some preliminary results as follows.

Proposition 3.1. *For any two resource allocations, say* $\mathbf{r}_i$ *and* $\mathbf{r}_j$*, if* $r_{i1} > r_{j1}$*, then we have* $r_{i2} < r_{j2}$.

Proof. We use proof by contradiction. Suppose $r_{i1} > r_{j1}$ and $r_{i2} \geq r_{j2}$. Then it means that for resource allocation $\mathbf{r}_j$, the remaining (unallocated) resources are sufficient to accommodate at least one task from user 1. This contradicts the fact that all resource allocations considered satisfy the Saturation Property.

This complete the proof. □

3.3 Case I and Solution

Without loss of generality, assume $\mathcal{G}_2 = \Phi$. So $\forall \mathbf{r}_j \in \mathcal{R}$, we have $r_{j1} \geq r_{j2}$. Consider K resource allocations $\mathbf{r}_1, \mathbf{r}_2, ..., \mathbf{r}_K$ from $\mathcal{R}$, and the K resource allocations are ordered such that $r_{11} < r_{21} < ... < r_{K1}$. Then from Proposition 3.1, we have $r_{12} > r_{22} > ... > r_{K2}$. Further, since $r_{11} \geq r_{12}$, we have $r_{K1} > ... > r_{21} > r_{11} \geq r_{12} > r_{22} > ... > r_{K2}$, which implies that among all resource allocations $\mathbf{r}_1, \mathbf{r}_2, ..., \mathbf{r}_K$, resource allocation $\mathbf{r}_1$ has the smallest difference between its two elements.

We have the following proposition.

Proposition 3.2. *Consider K resource allocations $\mathbf{r}_1, \mathbf{r}_2, ..., \mathbf{r}_K \in \mathcal{R}$ such that 1) $\forall \mathbf{r}_j$, we have $r_{j1} \geq r_{j2}$ and 2) $r_{11} < r_{21} < ... < r_{K1}$. Then to maximize $\min(\sum_{j=1}^{K} r_{j1}t_j, \sum_{j=1}^{K} r_{j2}t_j)$, we have $t_1 = 1, t_2 = t_3 = ... = t_K = 0$.*

Proof. Since $\forall \mathbf{r}_j$, we have $r_{j1} \geq r_{j2}$, then we have $\min(\sum_{j=1}^{K} r_{j1}t_j, \sum_{j=1}^{K} r_{j2}t_j) = \sum_{j=1}^{K} r_{j2}t_j$. Since $r_{11} < r_{21} < ... < r_{K1}$, from Proposition 3.1 we have $r_{12} > r_{22} > ... > r_{K2}$.

We use proof by contradiction. Assume that when $\min(\sum_{j=1}^{K} r_{j1}t_j, \sum_{j=1}^{K} r_{j2}t_j)$ is maximized, the optimal values of $t_1, t_2, ..., t_K$ are given as $t_1^*, t_2^*, ..., t_K^*$, and there exists $l \in \{2, 3, ..., K\}$ such that $t_l^* > 0$. So the maximal value of the objective function $\min(\sum_{j=1}^{K} r_{j1}t_j, \sum_{j=1}^{K} r_{j2}t_j)$ is given as $\sum_{j=1}^{K} r_{j2}t_j^*$.

Consider another set of the values of $t_1, t_2, ..., t_K$, given as $t_1^\dagger, t_2^\dagger, ..., t_K^\dagger$ such that $t_1^\dagger = t_1^* + t_l^*, t_l^\dagger = 0$, and $t_v^\dagger = t_v^*$ for $v \in \{1, 2, ..., K\}, v \neq 1, v \neq l$. Then the achieved objective function is given as

$$\sum_{j=1}^{K} r_{j2}t_j^\dagger = \sum_{j=1}^{K} r_{j2}t_j^* + r_{12}t_l^* - r_{l2}t_l^* > \sum_{j=1}^{K} r_{j2}t_j^*$$

where the inequality comes from the fact that $r_{12} > r_{22} > ... > r_{K2}$. This contradicts the fact that $t_1^*, t_2^*, ..., t_K^*$ achieve the maximal objective function.

This completes the proof. □

Then, it can be concluded that, for Case I, the optimal solution of Problem (3.1) is: there is only one time slot, which implements the resource allocation that has the smallest difference between its two elements. Further, for Case I (i.e., in any resource allocation in $\mathcal{R}$, the dominant resource share of a user is always higher than or equal to that of the other user), the resource allocation that has the smallest difference between its two elements is apparently the DRF allocation. Therefore, for Case I, a single time slot using the DRF allocation is optimal.

3.4 Case II and Solution

In Case II, Problem (3.1) becomes

$$\begin{aligned} \max_{K,\mathbf{r}_1,\mathbf{r}_2,...,\mathbf{r}_K\in\mathcal{G}_1\cup\mathcal{G}_2,t_1,t_2,...,t_K} \quad & \min(\sum_{j=1}^{K} r_{j1}t_j, \sum_{j=1}^{K} r_{j2}t_j) \\ \text{subject to:} \quad & t_1+t_2+...+t_K=1. \end{aligned} \tag{3.2}$$

We have the following proposition for Case II

Proposition 3.3. *For Case II, to achieve optimality in Problem (3.2), at least one time slot should implement a resource allocation from set $\mathcal{G}_1$, and at least one time slot should implement a resource allocation from set $\mathcal{G}_2$.*

Proof. We use proof by contradiction. Assume that in an optimal solution of Problem (3.2), all time slots implement resource allocations from set $\mathcal{G}_1$. Then from proof of proposition 3.2, the optimal solution of Problem (3.2) only has one time slot. Denote the resource allocation implemented in the time slot as $\mathbf{r}_1 = [r_{11}, r_{12}]^T \in \mathcal{G}_1$. Since $\mathbf{r}_1 \in \mathcal{G}_1$, we have $r_{11} > r_{12}$. Then the optimal objective function value of Problem (3.2) is given as r_{12}.

We randomly pick up a resource allocation from set $\mathcal{G}_2$, denoted as $\mathbf{r}_2 = [r_{21}, r_{22}]^T$, in which $r_{21} < r_{22}$. We should have $r_{22} > r_{12}$ (because otherwise, we have $r_{11} > r_{12} \geq r_{22} > r_{21}$, which implies that resource allocation $\mathbf{r}_2$ does not satisfy the Saturation Property). Due to the same reasoning, we have $r_{11} > r_{21}$.

Now consider we have two time slots with duration $t_1 = 1 - \delta$ and $t_2 = \delta$. Here δ is any positive value satisfying $0 < \delta < \frac{r_{11}-r_{12}}{(r_{11}-r_{12})+(r_{22}-r_{21})}$. Resource allocations $\mathbf{r}_1$ and $\mathbf{r}_2$ are implemented in the two slots, respectively. Then user 1's average dominant resource share over the two slots is given as $r_{11} - (r_{11} - r_{21})\delta$, which is less than r_{11}, while user 2's average dominant resource share over the two slots is given as $r_{12} + (r_{22} - r_{12})\delta$, which is more than r_{12}.

The difference of the two users' average dominant resource shares is given as

$$\begin{aligned} & r_{11} - (r_{11} - r_{21})\delta - [r_{12} + (r_{22} - r_{12})\delta] \\ = \quad & r_{11} - r_{12} - [(r_{11} - r_{12}) + (r_{22} - r_{21})]\delta \\ > \quad & r_{11} - r_{12} - [(r_{11} - r_{12}) + (r_{22} - r_{21})]\tfrac{r_{11}-r_{12}}{(r_{11}-r_{12})+(r_{22}-r_{21})} = 0 \end{aligned} \tag{3.3}$$

which means that user 2 has less average dominant resource share over the two slots. So the objective function of Problem (3.2) over the two slots is $r_{12} + (r_{22} - r_{12})\delta$, which is more than r_{12}, the optimal objective function value of Problem (3.2). Thus, there is a contradiction.

This completes the proof. □

Based on the results in Proposition 3.3, we have the following proposition in order.

Proposition 3.4. *In Case II, when optimality of Problem (3.2) is achieved, the average dominant resource share of the two users are equal.*

Proof. We use proof by contradiction. Assume in an optimal time-resource allocation of Problem (3.2), the two users have different average dominant resource share. Without loss of generality, assume user 1 and user 2 have average dominant resource share A_1 and A_2, respectively, with $A_1 > A_2$. Thus, the optimal objective function value of Problem (3.2) is A_2.

From Proposition 3.3, when the optimality is achieved, two resource allocations from $\mathcal{G}_1$ and $\mathcal{G}_2$, respectively, should be implemented in two slots. Assume $\mathbf{r}_1 = [r_{11}, r_{12}]^T \in \mathcal{G}_1$ and $\mathbf{r}_2 = [r_{21}, r_{22}]^T \in \mathcal{G}_2$ are implemented in slot 1 with duration

t_1 and slot 2 with duration t_2, respectively. So we have $r_{11} > r_{12}$ and $r_{21} < r_{22}$. Similar to proof of Proposition 3.3, we should have $r_{11} > r_{21}$ and $r_{22} > r_{12}$.

We do the following change to the optimal time-resource allocation: duration of slot 1 changes to $t_1 - \delta$, and duration of slot 2 changes to $1+\delta$. Here δ is any positive value satisfying $0 < \delta < \frac{A_1 - A_2}{(r_{11}-r_{12})+(r_{22}-r_{21})}$. In the new time-resource allocation, user 1 has average dominant resource share $A_1 - (r_{11} - r_{21})\delta$, which is less than A_1, while user 2 has average dominant resource share $A_2 + (r_{22} - r_{12})\delta$, which is more than A_2.

In the new time-resource allocation, the difference of the two users' average dominant resource shares is given as

$$\begin{aligned} & A_1 - (r_{11} - r_{21})\delta - [A_2 + (r_{22} - r_{12})\delta] \\ = \quad & A_1 - A_2 - [(r_{11} - r_{12}) + (r_{22} - r_{21})]\delta \\ > \quad & A_1 - A_2 - [(r_{11} - r_{12}) + (r_{22} - r_{21})]\frac{A_1 - A_2}{(r_{11}-r_{12})+(r_{22}-r_{21})} = 0 \end{aligned} \tag{3.4}$$

which means that user 2 has less average dominant resource share. So for Problem (3.2), the new time-resource allocation has objective function value being $A_2 + (r_{22} - r_{12})\delta$, which is more than A_2, the optimal objective function value of Problem (3.2). Thus, there is a contradiction.

This completes the proof. □

Proposition 3.4 means that Problem (3.2) is equivalent to the following problem:

$$\begin{aligned} \max_{K, \mathbf{r}_1, \mathbf{r}_2, ..., \mathbf{r}_K \in \mathcal{G}_1 \cup \mathcal{G}_2, t_1, t_2, ..., t_K} \quad & \sum_{j=1}^{K} r_{j1} t_j \\ \text{subject to:} \quad & \sum_{j=1}^{K} r_{j1} t_j = \sum_{j=1}^{K} r_{j2} t_j; \\ & t_1 + t_2 + ... + t_K = 1. \end{aligned} \tag{3.5}$$

To solve Problem (3.5), we add additional constraint $K = 2$, $K = 3$, and $K \geq 4$, and solve the problem accordingly in the subsequent subsections. Note that optimal performance (in terms of the objective function) of Problem (3.5) with

additional constraint $K = l$ is not better than that of Problem (3.5) with additional constraint $K = l + v$ for any $v > 0$.

3.4.1 When K is fixed to 2

When K is fixed to 2, we have two time slots, slot 1 with duration t_1 and slot 2 with duration t_2. From Proposition 3.3, The two slots implement resource allocation $\mathbf{r}_1 = [r_{11}, r_{12}]^T \in \mathcal{G}_1$ and $\mathbf{r}_2 = [r_{21}, r_{22}]^T \in \mathcal{G}_2$, respectively. So we have $r_{11} > r_{12}$ and $r_{21} < r_{22}$. Further, similar to proof of Proposition 3.3, we should have $r_{11} > r_{21}$ and $r_{22} > r_{12}$.

Then Problem (3.5) simplifies to

$$\begin{aligned} \max_{\mathbf{r}_1 \in \mathcal{G}_1, \mathbf{r}_2 \in \mathcal{G}_2, 0<t_1<1} \quad & r_{11}t_1 + r_{21}(1-t_1) \\ \text{subject to:} \quad & r_{11}t_1 + r_{21}(1-t_1) = r_{12}t_1 + r_{22}(1-t_1). \end{aligned} \tag{3.6}$$

For Problem (3.6), we may have $|\mathcal{G}_1| \cdot |\mathcal{G}_2|$ pairs of $\mathbf{r}_1$ and $\mathbf{r}_2$. For each pair $\mathbf{r}_1 = [r_{11}, r_{12}]^T$ and $\mathbf{r}_2 = [r_{21}, r_{22}]^T$, the constraint of Problem (3.6) determines that the value of t_1 is given as $t_1 = \frac{r_{22}-r_{21}}{(r_{11}-r_{12})+(r_{22}-r_{21})}$, and thus, the objective function value is given in closed-form as $r_{11}\frac{r_{22}-r_{21}}{(r_{11}-r_{12})+(r_{22}-r_{21})} + r_{21}\frac{r_{11}-r_{12}}{(r_{11}-r_{12})+(r_{22}-r_{21})}$. Since we have $|\mathcal{G}_1| \cdot |\mathcal{G}_2|$ pairs of $\mathbf{r}_1$ and $\mathbf{r}_2$, we have $|\mathcal{G}_1| \cdot |\mathcal{G}_2|$ objective function values. Then we can select the maximal objective function value, and the corresponding $\mathbf{r}_1$, $\mathbf{r}_2$, and t_1 are an optimal time-resource allocation of Problem (3.6).

3.4.2 When K is fixed to 3

When K is fixed to 3, we have three time slots, slot 1, slot 2, and slot 3 with duration t_1, t_2 and t_3, respectively. The resource allocations in the three slots are $\mathbf{r}_1 = [r_{11}, r_{12}]^T$, $\mathbf{r}_2 = [r_{21}, r_{22}]^T$, and $\mathbf{r}_3 = [r_{31}, r_{32}]^T$, respectively. From Proposition 3.3, assume $\mathbf{r}_1 \in \mathcal{G}_1$, $\mathbf{r}_2 \in \mathcal{G}_2$, and $\mathbf{r}_3 \in \mathcal{G}_1 \cup \mathcal{G}_2$. So we have $r_{11} > r_{12}$ and $r_{21} < r_{22}$.

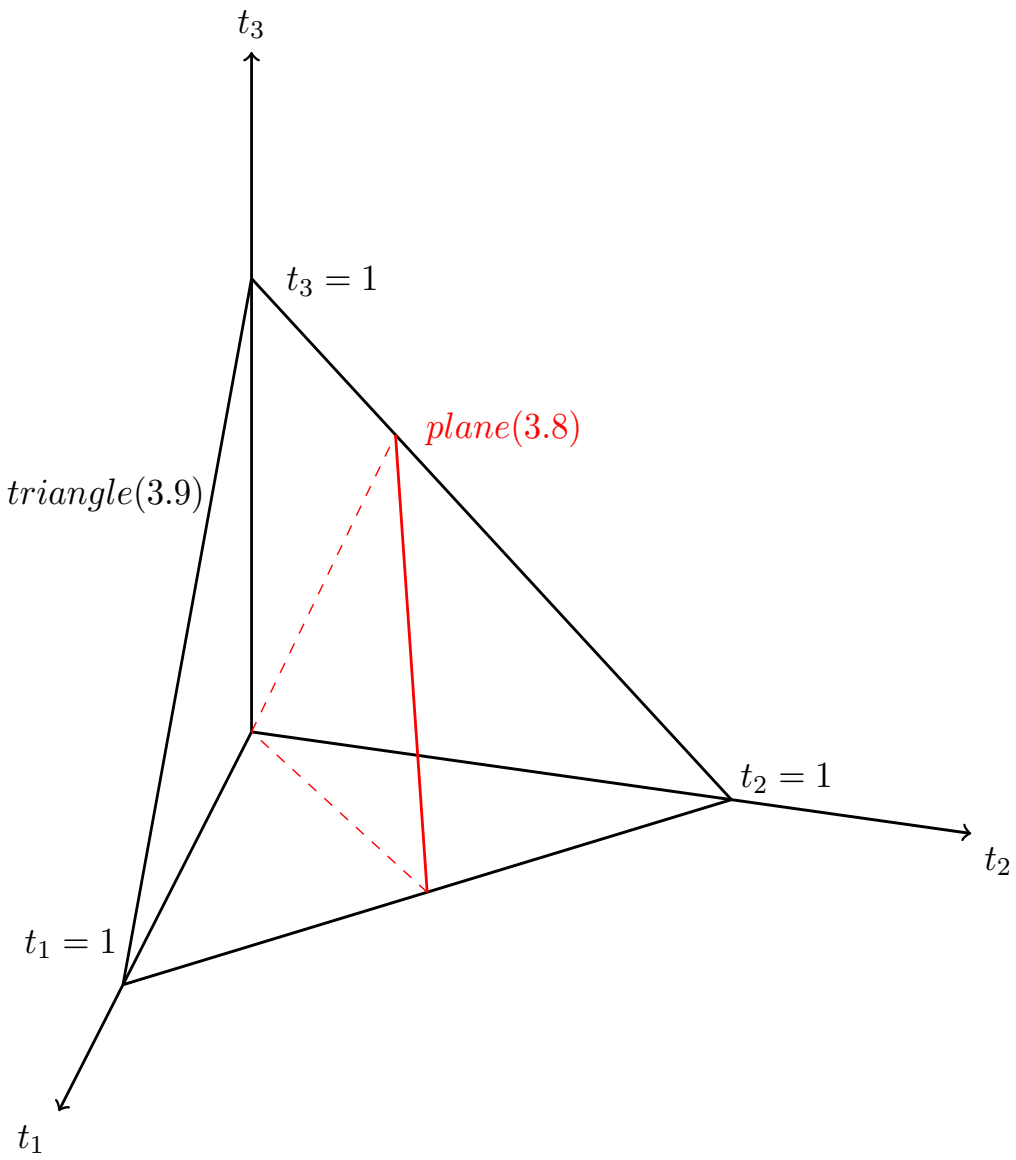

Fig. 3.1. triangle (3.9), plane (3.8), and their common line segment

Then Problem (3.5) simplifies to

$$\max_{\mathbf{r}_1\in\mathcal{G}_1,\mathbf{r}_2\in\mathcal{G}_2,\mathbf{r}_3\in\mathcal{G}_1\cup\mathcal{G}_2,t_1,t_2,t_3} \quad r_{11}t_1 + r_{21}t_2 + r_{31}t_3 \tag{3.7}$$

$$\text{subject to:} \quad r_{11}t_1 + r_{21}t_2 + r_{31}t_3 = r_{12}t_1 + r_{22}t_2 + r_{32}t_3 \tag{3.8}$$

$$t_1 + t_2 + t_3 = 1 \tag{3.9}$$

In a three-dimension space of (t_1, t_2, t_3), the constraint (3.9) is a triangle formed by three points $(0, 0, 1), (0, 1, 0), (1, 0, 0)$, as shown in Fig.3.1. Constraint (3.8) means a plane. For constraints (3.8) and (3.9), $t_1 = \frac{r_{22}-r_{21}}{(r_{11}-r_{12})+(r_{22}-r_{21})}$, $t_2 = \frac{r_{11}-r_{12}}{(r_{11}-r_{12})+(r_{22}-r_{21})}$, $t_3 = 0$ can satisfy both of them, which means point

$$\left(\frac{r_{22}-r_{21}}{(r_{11}-r_{12})+(r_{22}-r_{21})}, \frac{r_{11}-r_{12}}{(r_{11}-r_{12})+(r_{22}-r_{21})}, 0\right)$$

is a common point of the triangle (3.9) and the plane (3.8), which implies that the triangle and the plane cross each other, resulting in a common line segment of them. An example of the line segment is shown in Fig.3.1. Any point on the line segment satisfies (3.8) and (3.9). Thus, we need to find the point on the line segment that maximizes the objective function $r_{11}t_1+r_{21}t_2+r_{31}t_3$. From constraints (3.8) and (3.9), both t_2 and t_3 can be expressed as a linear function of t_1. Thus, the objective function can be expressed as a linear function of t_1, denoted $f(t_1)$. Apparently, along the line segment, $f(t_1)$ achieves its maximal value at one end point. At either end point, we have $t_1t_2t_3 = 0$. This means that in the optimal time-resource allocation of Problem (3.7), one from t_1, t_2, t_3 should be zero. In other words, Problem (3.5) with additional constraint $K = 3$ simplifies to Problem (3.5) with additional constraint $K = 2$, which has been solved in Section 3.4.1.

3.4.3 When K is fixed to a value more than 4

We have the following proposition.

Proposition 3.5. *Problem (3.5) with additional constraint $K = v$ ($v \in \{4, 5, ...\}$) simplifies to Problem (3.5) with additional constraint $K = 2$.*

Proof. We use mathematical induction to prove.

First consider $K = 4$. In an optimal time-resource allocation, suppose the four slots have duration t_1, t_2, t_3, t_4, and resource allocations $\mathbf{r}_1, \mathbf{r}_2, \mathbf{r}_3, \mathbf{r}_4$, respectively. From Proposition 3.3, at least one resource allocation should be from $\mathcal{G}_1$, and at least one resource allocation should be from $\mathcal{G}_2$. So we assume $\mathbf{r}_1 \in \mathcal{G}_1$ and $\mathbf{r}_2 \in \mathcal{G}_2$.

The two users' dominant resource share in the first two slots are $r_{11}t_1 + r_{21}t_2$ and $r_{12}t_1 + r_{22}t_2$, respectively.

- Scenario 1: If the two users have the same dominant resource share in the first two slots, i.e., $r_{11}t_1 + r_{21}t_2 = r_{12}t_1 + r_{22}t_2$, then they also have the same dominant resource share in the last two slots, i.e., $r_{31}t_3 + r_{41}t_4 = r_{32}t_3 + r_{42}t_4$. Then we should have $(r_{11}t_1 + r_{21}t_2)\frac{1}{t_1+t_2} = (r_{31}t_3 + r_{41}t_4)\frac{1}{t_3+t_4}$. This is because we have a contradiction if the equality does not hold, as follows. If $(r_{11}t_1 + r_{21}t_2)\frac{1}{t_1+t_2} > (r_{31}t_3 + r_{41}t_4)\frac{1}{t_3+t_4}$. Then the system has a better objective function if the time allocation is changed to $\frac{t_1}{t_1+t_2}, \frac{t_2}{t_1+t_2}, 0, 0$ for the four time slots. If $(r_{11}t_1 + r_{21}t_2)\frac{1}{t_1+t_2} < (r_{31}t_3 + r_{41}t_4)\frac{1}{t_3+t_4}$. Then the system has a better objective function if the time allocation is changed to $0, 0, \frac{t_3}{t_3+t_4}, \frac{t_4}{t_3+t_4}$ for the four time slots.

  Since $(r_{11}t_1 + r_{21}t_2)\frac{1}{t_1+t_2} = (r_{31}t_3 + r_{41}t_4)\frac{1}{t_3+t_4}$, then the optimal objective function value of Problem (3.5) with $K = 4$ can also be achieved by using two slots with duration $\frac{t_1}{t_1+t_2}$ and $\frac{t_2}{t_1+t_2}$, and resource allocation $\mathbf{r}_1$ and $\mathbf{r}_2$, respectively.

- Scenario 2: If the two users have the different dominant resource share in the first two slots, i.e., $r_{11}t_1 + r_{21}t_2 \neq r_{12}t_1 + r_{22}t_2$. Without loss of generality, assume $r_{11}t_1 + r_{21}t_2 > r_{12}t_1 + r_{22}t_2$, which leads to $(r_{11} - r_{12})t_1 > (r_{22} - r_{21})t_2$. Define $t_1^\dagger$ such that $(r_{11} - r_{12})t_1^\dagger = (r_{22} - r_{21})t_2$. Then the optimal time-resource allocation for the four time slots can be virtually viewed as two portions: the first portion includes two slots with durations $t_1^\dagger$ and t_2 and resource allocations $\mathbf{r}_1$ and $\mathbf{r}_2$, and the second portion includes three slots with durations $t_3, t_4, t_1 - t_1^\dagger$ and resource allocations $\mathbf{r}_3, \mathbf{r}_4, \mathbf{r}_1$. In the first

portion, the two users have the same dominant resource share, and thus, they also have the same dominant resource share in the second portion. Then similar to the reasoning in Scenario 1, we should have $(r_{11}t_1^\dagger + r_{21}t_2)\frac{1}{t_1^\dagger+t_2} = (r_{31}t_3+r_{41}t_4+r_{11}(t_1-t_1^\dagger))\frac{1}{t_3+t_4+t_1-t_1^\dagger}$, and thus, the optimal objective function value of Problem (3.5) with $K = 4$ can also be achieved by using two slots with duration $\frac{t_1^\dagger}{t_1^\dagger+t_2}$ and $\frac{t_2}{t_1^\dagger+t_2}$, and resource allocation $\mathbf{r}_1$ and $\mathbf{r}_2$, respectively.

Thus, Proposition 3.5 holds for $K = 4$.

Now we assume that Proposition 3.5 holds for $K = l$ ($l \geq 4$). We need to prove that Proposition 3.5 also holds for $K = l + 1$.

For $K = l + 1$, we have $l + 1$ slots with durations $t_1, t_2, ..., t_{l+1}$ and resource allocations $\mathbf{r}_1, \mathbf{r}_2, ..., \mathbf{r}_{l+1}$. From Proposition 3.3, we assume $\mathbf{r}_1 \in \mathcal{G}_1, \mathbf{r}_2 \in \mathcal{G}_2$.

The two users' dominant resource share in the first two slots are $r_{11}t_1 + r_{21}t_2$ and $r_{12}t_1 + r_{22}t_2$, respectively.

- If the two users have the same dominant resource share in the first two slots, i.e., $r_{11}t_1 + r_{21}t_2 = r_{12}t_1 + r_{22}t_2$, then they also have the same dominant resource share in the last $l - 1$ slots. According to the induction hypothesis, the optimal time-resource allocation in the second portion (which has $l - 1$ slots) is not better than the optimal time-resource allocation with two slots. Thus, overall, the optimal time-resource allocation with $l+1$ slots is not better than the optimal time-resource allocation with four slots, which is not better than the optimal time-resource allocation with two slots.

- If the two users have different dominant resource share in the first two slots, i.e., $r_{11}t_1 + r_{21}t_2 \neq r_{12}t_1 + r_{22}t_2$. Without loss of generality, assume $r_{11}t_1 + r_{21}t_2 > r_{12}t_1 + r_{22}t_2$, which leads to $(r_{11} - r_{12})t_1 > (r_{22} - r_{21})t_2$. Define $t_1^\dagger$ such that $(r_{11}-r_{12})t_1^\dagger = (r_{22}-r_{21})t_2$. Then the optimal time-resource allocation for the $l+1$ slots can be virtually viewed as two portions: the first portion includes two slots with durations $t_1^\dagger$ and t_2 and resource allocations $\mathbf{r}_1$ and $\mathbf{r}_2$, and the second portion includes l slots with durations $t_3, t_4, ..., t_{l+1}, t_1-t_1^\dagger$ and resource allocations $\mathbf{r}_3, \mathbf{r}_4, ..., \mathbf{r}_{l+1}, \mathbf{r}_1$. In the first portion, the two users have

the same dominant resource share, and thus, they also have the same dominant resource share in the second portion. According to the induction hypothesis, the optimal time-resource allocation of the second portion (which has l slots) is not better than the optimal time-resource allocation with two slots. Thus, overall, the optimal time-resource allocation with $l+1$ slots is not better than the optimal time-resource allocation with four slots, which is not better than the optimal time-resource allocation with two slots.

Thus, the optimal time-resource allocation with $l+1$ slots is not better than the optimal time-resource allocation with two slots. On the other hand, the optimal time-resource allocation with $l+1$ slots is always not worse than the optimal time-resource allocation with two slots. Therefore, Problem (3.5) with additional constraint $K = l+1$ simplifies to Problem (3.5) with additional constraint $K = 2$.

This completes the proof. □

3.5 Case III and Solution

In Case III, set $\mathcal{G}_3$ is non-empty. Then based on the Saturation Property, $\mathcal{G}_3$ only includes one resource allocation, which is the DRF allocation. Then theoretically, an optimal solution to Problem (3.1) includes two portions: the first portion has one slot with duration τ and with DRF as the resource allocation, and the second portion with total duration $1-\tau$ has a number of slots with resource allocations from $\mathcal{G}_1 \cup \mathcal{G}_2$. The two users have the same dominant resource share in the first portion since $\mathcal{G}_3$ includes resource allocation in which the two users have the same dominant resource share. This also means that the two users have the same dominant resource share in the second portion. Thus, to solve Problem (3.1), for any specific τ, we should find the optimal solution to maximize the equal dominant resource share of the two users in the second portion, which is equivalent to Problem (3.2) and is solved in Section 3.4.

Since theoretically, an optimal solution to Problem (3.1) includes two portions, and the two users have equal dominant resource share in each portion, the portion

whose normalized (with respect to the duration of the portion) objective function is smaller should have duration zero. In specific, we should first find the optimal solution to Problem (3.2). If its optimal objective function (which is the equal dominant resource share of the two users) is higher than the two users' equal dominant resource share in the DRF allocation, then optimal solution to Problem (3.2) is an optimal solution to Problem (3.1); otherwise, one time slot with DRF allocation is the optimal solution to Problem (3.1).

3.6 Overall Optimal Time-Resource Allocation Algorithm for Problem (3.1)

Since we have provided solutions for all the three cases of Problem (3.1) in Section 3.3 - 3.5, an overall optimal time-resource allocation algorithm for Problem (3.1) can be given in Algorithm 3.1.

Algorithm 3.1 Algorithm solving Problem (3.1).

1: Find all resource allocations, and categorize them into three sets $\mathcal{G}_1, \mathcal{G}_2, \mathcal{G}_3$.

2: If $\mathcal{G}_1 = \Phi$ (null set) or $\mathcal{G}_2 = \Phi$, then one slot with DRF allocation is the optimal solution, and proceed to Step 7.

3: Find $(\mathbf{r}_1^*, \mathbf{r}_2^*) = \arg\max_{\mathbf{r}_1 \in \mathcal{G}_1, \mathbf{r}_2 \in \mathcal{G}_2} r_{11} \frac{r_{22}-r_{21}}{(r_{11}-r_{12})+(r_{22}-r_{21})} + r_{21} \frac{r_{11}-r_{12}}{(r_{11}-r_{12})+(r_{22}-r_{21})}$.

4: If $\mathcal{G}_3 = \Phi$, then an optimal solution is to have two slots with $\mathbf{r}_1^*, \mathbf{r}_2^*$ as resource allocations and $t_1 = \frac{r_{22}-r_{21}}{(r_{11}-r_{12})+(r_{22}-r_{21})}, t_2 = \frac{r_{11}-r_{12}}{(r_{11}-r_{12})+(r_{22}-r_{21})}$, and proceed to Step 7.

5: If $r_{11}^* \frac{r_{22}^*-r_{21}^*}{(r_{11}^*-r_{12}^*)+(r_{22}^*-r_{21}^*)} + r_{21}^* \frac{r_{11}^*-r_{12}^*}{(r_{11}^*-r_{12}^*)+(r_{22}^*-r_{21}^*)}$ is higher than the minimum dominant resource share in the DRF allocation, then an optimal solution is to have two slots with $\mathbf{r}_1^*, \mathbf{r}_2^*$ as resource allocations and $t_1 = \frac{r_{22}-r_{21}}{(r_{11}-r_{12})+(r_{22}-r_{21})}, t_2 = \frac{r_{11}-r_{12}}{(r_{11}-r_{12})+(r_{22}-r_{21})}$, and proceed to Step 7.

6: One slot with the DRF allocation is optimal.

7: Exit.

Now we analyze the complexity of the proposed Algorithm 3.1. Denote the

number of resource allocations in $\mathcal{R}$ as N. It can be seen that the complexity of the proposed Algorithm 3.1 is determined by the complexity in Step 3 of Algorithm 3.1. Step 3 of Algorithm 3.1 has a complexity of $O(|\mathcal{G}_1| \cdot |\mathcal{G}_2|)$. Since $|\mathcal{G}_1| + |\mathcal{G}_2| \leq N$, an upper bound of $|\mathcal{G}_1| \cdot |\mathcal{G}_2|$ is given as $N^2/4$. Thus, the worst-case complexity of Algorithm 3.1 is expressed as $O(N^2)$.

As a comparison, we consider complexity of solving Problem (3.1) by using an exhaustive search. Totally there are 2^N possible combinations of the resource allocations used in the time-resource allocation. Thus, the complexity of an exhaustive search is expressed as $O(2^N)$, which is much higher than that of our proposed Algorithm 3.1.

3.7 Conclusion

In this chapter, based on the idea of time division resource allocation, we formulate an average dominant resource share based time-resource allocation problem. In general, having more time slots should have performance not worse than that with fewer time slots. However, interestingly, it is proved in this chapter that the optimal time-resource allocation performance can be achieved with up to two time slots.

Chapter 4

Theoretical Performance Bounds for the Two Users

In Chapter 3, we have developed an algorithm to find the optimal performance of two users. Now we answer the following question: are there theoretical performance bounds for the two users?

In Chapter 3, the system can accommodate an *integer* number of tasks from each user. Therefore, for a resource allocation, it is possible that none of the resource types are completely consumed. Since our target now is at the theoretical bounds, we relax the integer requirements on the number of tasks from a user. Rather, we consider an ideal case that the number of tasks from a user can be any continuous value. We derive theoretical bounds in the ideal case.

For presentation simplicity, the capacity of each resource type is normalized to 1. We first define a resource factor for user $u \in \{1, 2\}$ in resource type $r \in \{1, 2, ..., m\}$: $q_{r,u} = \frac{\text{Demand of resource } r \text{ from a task of user } u}{\text{Demand of dominant resource from a task of user } u}$. Therefore, for a user, the resource factor for dominant resource is $q_{r,u} = 1$; and for non-dominant resource, we have $q_{r,u} \leq 1$. We also define the resource type $\arg\max_r(q_{r,1} + q_{r,2})$ as the *most popular resource*. Also define $Q \triangleq \max_r(q_{r,1} + q_{r,2})$.

Assume the total time duration is 1. The total duration is divided into a number of time slots, and in each time slot, a resource allocation is implemented. Since the number of tasks from a user can be a continuous value, for a resource allocation,

at least one resource type should be consumed completely. Denote $x_1(t)$ and $x_2(t)$ ($0 \leq t \leq 1$) as the amount of dominant resource share of user 1 and user 2 over the slots.

In the time-resource allocation, we make the two users have equal average dominant resource share, and try to maximize their equal average dominant resource share, referred to as *optimal average dominant resource share (ADRS)*.

If the two users have the same dominant resource type, the following lemma is in order.

Lemma 4.1. If the two users have the same dominant resource type, the optimal ADRS of the two users is equal to $1/2$.

Proof. Without loss of generality, assume resource type 1 is the dominant resource for both users. So we have $q_{1,1} = q_{1,2} = 1$. Then apparently, for each resource allocation, resource type 1 is completely consumed. Thus, for any $t \in [0, 1]$, we have

$$x_1(t) + x_2(t) = 1. \tag{4.1}$$

The ADRS of user 1 and user 2 is given as $\int_0^1 x_1(t)dt$ and $\int_0^1 x_2(t)dt$, respectively. Since we make the two users have equal ADRS, we have

$$\int_0^1 x_1(t)dt = \int_0^1 x_2(t)dt. \tag{4.2}$$

From (4.1) and (4.2), we have $\int_0^1 x_1(t)dt = \int_0^1 x_2(t)dt = 1/2$.

This completes the proof. □

If the two users have different dominant resource type, the following lemma is in order.

Lemma 4.2. If the two users have different dominant resource type, the optimal ADRS of the two users is equal to $\frac{1}{Q}$. When the optimal ADRS is achieved, the most popular resource is completely consumed in all resource allocations over all time slots.

Proof. For presentation simplicity, here we consider two resource types: resource 1 and resource 2. But the proof can be straightforwardly extended to the case with more than two resource types.

Assume the dominant resource type for user 1 and user 2 are resource 1 and resource 2, respectively. So we have $q_{1,1} = q_{2,2} = 1$, and $q_{2,1} \leq 1, q_{1,2} \leq 1$. Assume the most popular resource is resource 1. So we have $Q = q_{1,1} + q_{1,2}$.

Recall that in each resource allocation, one resource type should be completely consumed. Without loss of generality, assume that for duration $t \in [0, \tau]$, all resource allocations completely consume resource 2, and for duration $t \in (\tau, 1]$, all resource allocations completely consume resource 1.

For duration $t \in [0, \tau]$, resource 1 is not completely consumed. Denote $w(t)$ $(t \in [0, \tau])$ as the amount of resource 1 that is not consumed. Therefore, we have

$$q_{1,1}x_1(t) + q_{1,2}x_2(t) = 1 - w(t)$$

which leads to

$$q_{1,1}\int_0^\tau x_1(t)dt + q_{1,2}\int_0^\tau x_2(t)dt = \tau - \int_0^\tau w(t)dt. \tag{4.3}$$

For duration $t \in (\tau, 1]$, resource 1 is completely consumed. So we have

$$q_{1,1}x_1(t) + q_{1,2}x_2(t) = 1$$

which leads to

$$q_{1,1}\int_\tau^1 x_1(t)dt + q_{1,2}\int_\tau^1 x_2(t)dt = 1 - \tau. \tag{4.4}$$

Additionally, since the two users have the same ADRS, we have:

$$\begin{aligned} ADRS &= \int_0^\tau x_1(t)dt + \int_\tau^1 x_1(t)dt \\ &= \int_0^\tau x_2(t)dt + \int_\tau^1 x_2(t)dt. \end{aligned} \tag{4.5}$$

From (4.3), (4.4), and (4.5), we have

$$\begin{aligned} ADRS =& \int_0^{\tau} x_1(t)dt + \int_{\tau}^{1} x_1(t)dt \\ =& \frac{1}{q_{1,1}+q_{1,2}}(1 - \int_0^{\tau} w(t)dt) \\ =& \frac{1}{Q}(1 - \int_0^{\tau} w(t)dt). \end{aligned} \tag{4.6}$$

Thus, to maximize ADRS, we should set $\tau = 0$, which leads to $ADRS = \frac{1}{Q}$. Here $\tau = 0$ means that all resource allocations for duration $t \in [0, 1]$ should completely consume resource 1.

This completes the proof. □

When the two users have the same dominant resource type, we have $Q = 2$. So based on the above two lemmas, it can be concluded that the optimal ADRS of each user is given as $1/Q$, which can serve as a theoretical upper bound performance of the ADRS in time division allocation.

Chapter 5

Performance Evaluation

In this chapter, we will verify the performance of our proposed time division allocation (TDA) strategy.

We simulate two types of tasks from two users. User 1 is CPU dominant, and the demand of a task varies from $< 1\,CPU, 1\,GB >$ to $< 5\,CPU, 2\,GB >$. User 2 is memory dominant, and the demand of one task varies from $< 1\,CPU, 1\,GB >$ to $< 3\,CPU, 6\,GB >$. Totally we have 180 scenarios of the demand configuration of the two users.

The comparison of fairness in dominant resource is shown in Figure **??**. The x-axis represents the sequence number of the 180 scenarios. The y-axis represents the *dominant resource difference ratio*, which is the ratio of the absolute difference of the two users' average dominant resource shares to the minimum average dominant resource share of the two users. For our TDA strategy, the dominant resource difference ratio is always 0 for all the 180 scenarios, which means the two users achieve equal average dominant resource share. On the other hand, the dominant resource difference ratio for DRF allocation is high in many scenarios. Among the 180 scenarios, the dominant resource difference ratio is higher than 0.5 in 56 scenarios.

Figure 5.2 shows the minimum average dominant resource share of the two users in the 180 scenarios. It can be seen that in each scenario, the minimum average dominant resource share in our TDA strategy is always higher than or equal to

that in the DRF strategy. Among the 180 scenarios, the two strategies achieve the same performance in 16 scenarios, while our TDA strategy performs better in 164 scenarios.

Figure 5.2 also shows the theoretical bounds of average dominant resource share in the 180 scenarios. Among the 180 scenarios, the DRF strategy achieves the theoretical bounds in 11 scenarios, while our TDA strategy achieves the theoretical bounds in 55 scenarios.

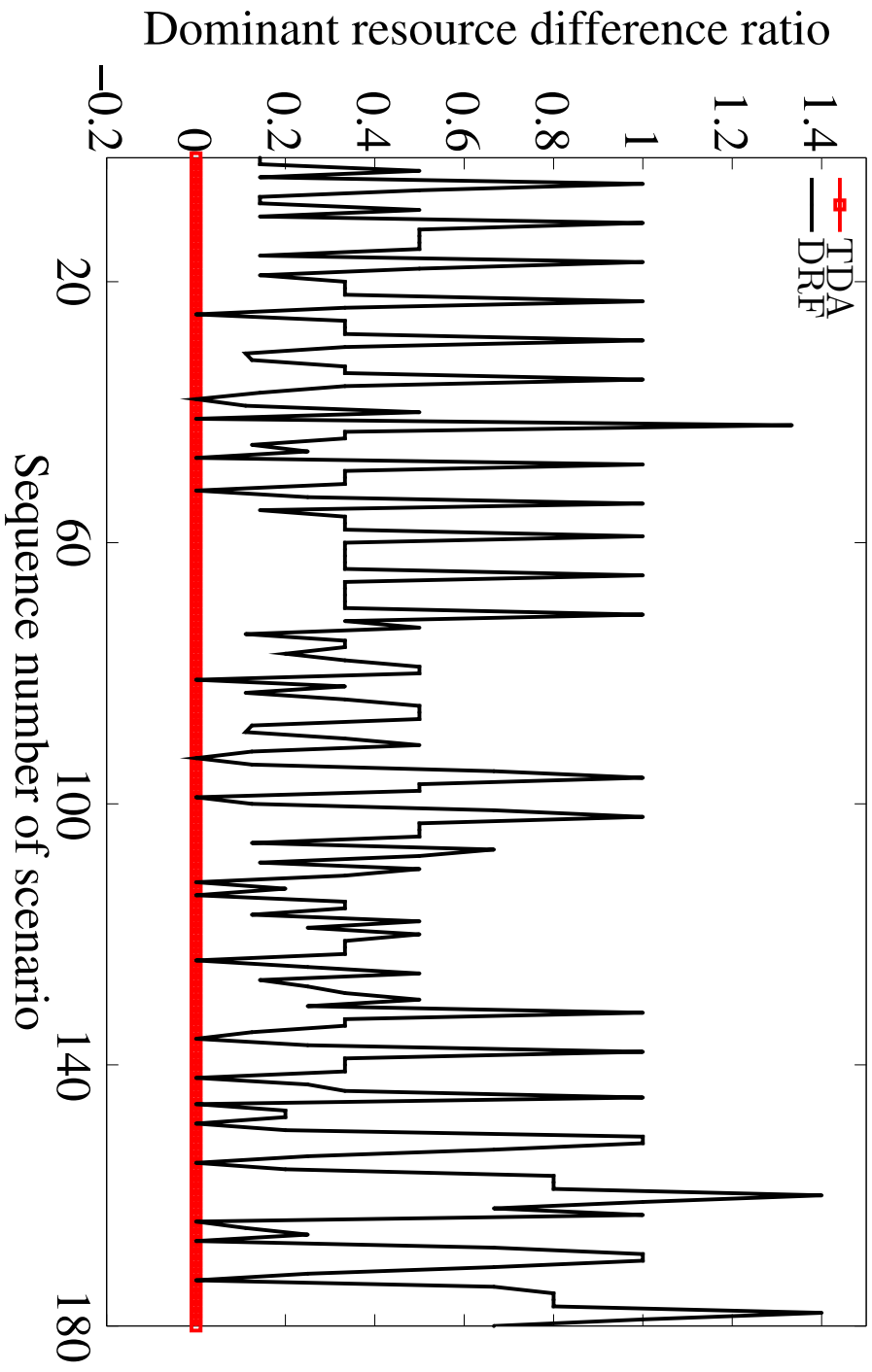

Fig. 5.1. Time Dominant resource difference ratio.

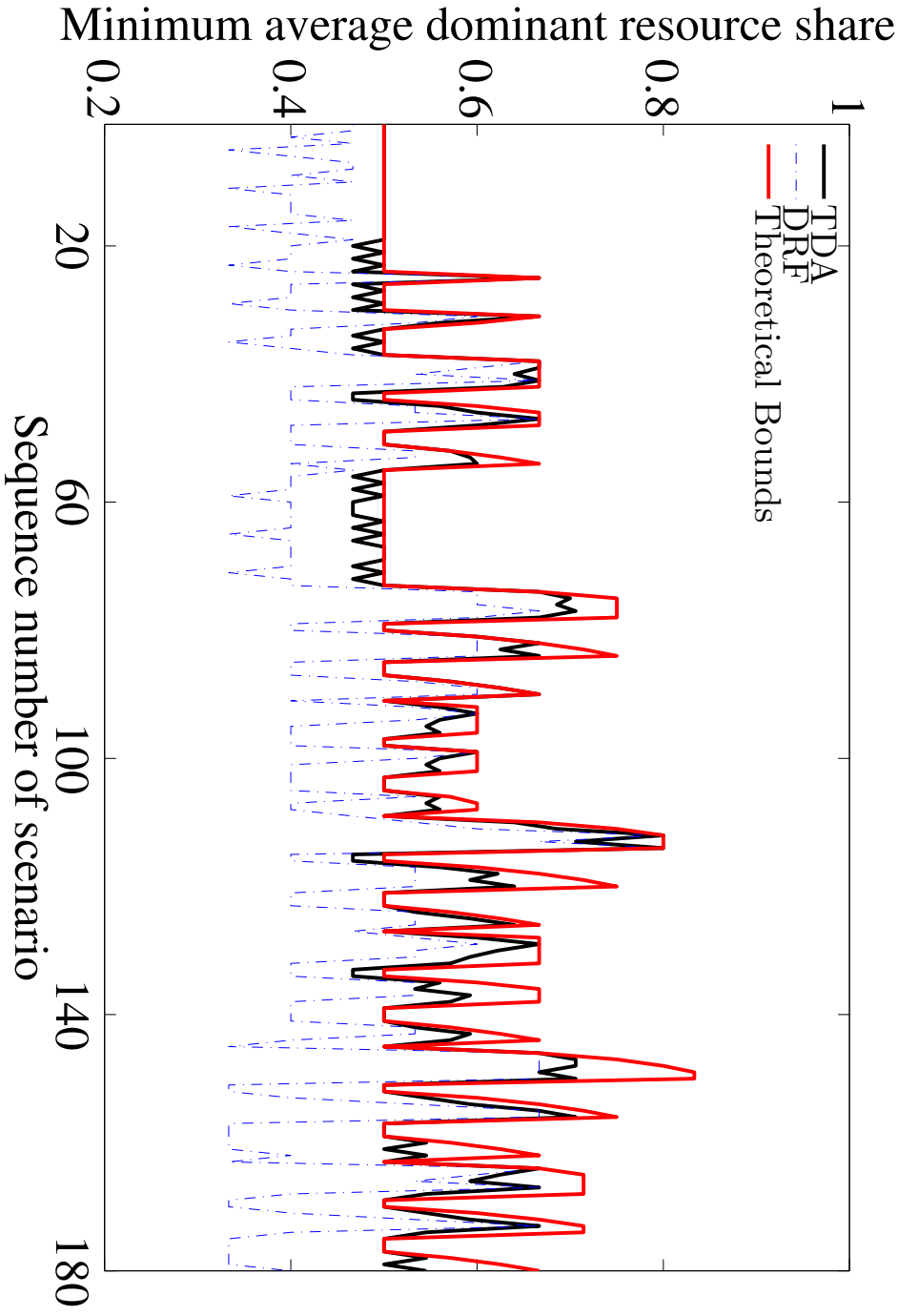

Fig. 5.2. Minimum average dominant resource share of the two users.

Chapter 6

Conclusion and Future Work

In this chapter, we will have a conclusion of this thesis and propose some work that can be done in the future.

6.1 Conclusion

This thesis discusses the resource allocation strategy in cloud environments. To have a fair allocation for heterogeneous users is the major challenge in this area. The default resource slot based allocation is neither fair nor efficient. The dominant resource fairness allocation aims to improve the fairness but the indivisibility of the task demand makes it hard to guarantee the two users have equal dominant resource share. In order to improve fairness as well as the utilization, we propose the time division dominant resource fairness allocation.

We first formulate a general time-resource allocation problem which tries to optimize the number of slots, the resource allocations over the slots, and the time allocation for the slots. Interestingly, we theoretically prove that the optimality can be achieved by using at most two slots. The theoretical bound performance for the time-resource allocation is also derived in this thesis.

The theoretical and simulation results show that time division allocation (TDA) is a very effective allocation scheduler for multiple-resource allocation in cloud environments.

6.2 Future Work

In this thesis, we have developed an algorithm for optimal time-resource allocation for two users. An interesting future work is to investigate the optimal time-resource allocation for more users. The following problems should be investigated: for N users, how many time slots are sufficient to achieve optimality? Is there a simple algorithm that can find the optimal solution quickly? Are there theoretical performance bounds? etc.

References

[1] M. Armbrust, *et al.*, "A view of cloud computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, Apr. 2010.

[2] E. Elghoneimy, O. Bouhali, and H. Alnuweiri, "Resource allocation and scheduling in cloud computing," in *Proceedings of 2012 International Conference on Computing, Networking and Communications (ICNC)*, pp. 309–314, 2012.

[3] Cloud Computing Wikipedia. http://en.wikipedia.org/wiki/Cloud_computing

[4] Amazon EC2. http://aws.amazon.com/ec2

[5] Apache, https://hadoop.apache.org/

[6] M. Isard, M. Budiu, Y. Yu, A. Birrell, D. Fetterly, "Dryad: Distributed data-parallel programs from sequential building blocks," in *Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems*, pp. 59–72, 2007.

[7] B. Hindman, *et al.* "Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center," in *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*, pp. 295–308, 2011.

[8] Mesos website. http://mesos.apache.org/

[9] S. Ghemawat, H. Gobioff, and S.-T. Leung. "The Google file system," *ACM SIGOPS Operating Systems Review*, vol. 37, no. 5, pp. 29–43, Dec. 2003.

[10] MapReduce Tutorial. http://hadoop.apache.org/docs/current/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html

[11] J. Dean and S. Ghemawat. "MapReduce: Simplified data processing on large clusters," *Communications of the ACM*, vol. 53, no. 1, pp. 107–113, Jan. 2008.

[12] M. Zaharia, *et al.* "Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling," in *Proceedings of the 5th European Conference on Computer Systems*, pp. 265–278, 2010.

[13] Hadoop Fair Scheduler Design Document. http://doc.mapr.com/display/MapR/Job+Scheduling

[14] A. Ghodsi, *et al.* "Dominant Resource Fairness: Fair Allocation of Multiple Resource Types," in *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation* , pp. 323–336, 2011.

[15] C. Delimitrou and C. Kozyrakis, "Quasar: Resource-efficient and QoS-aware cluster management," in *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 127–144, 2014.

[16] H. Liu, "A measurement study of server utilization in public clouds," in *Proceedings of IEEE Ninth International Conference on Dependable, Autonomic and Secure Computing(DASC)*, pp. 435–442, 2011.

[17] T. Sandholm and K. Lai. "MapReduce optimization using regulated dynamic prioritization," *ACM SIGMETRICS Performance Evaluation Review*, vol. 37, no. 1, pp. 299–310, Jan. 2009.

[18] D. C. Parkes, A. D. Procaccia, and N. Shah. "Beyond dominant resource fairness: extensions, limitations, and indivisibilities," in *Proceedings of the 13th ACM Conference on Electronic Commerce*, pp. 808–825, 2012.

[19] M. Isard, *et al.* “Quincy: Fair scheduling for distributed computing clusters,” in *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles*, pp. 261–276, 2009.

[20] Hadoop YARN. https://hadoop.apache.org/docs/r2.4.1/hadoop-yarn/hadoop-yarn-site/YARN.html

[21] YARN Background and Overview. http://hortonworks.com/blog/apache-hadoop-yarn-background-and-an-overview/

[22] T. Lan, D. Kao, M. Chiang, and A. Sabharwal, “An axiomatic theory of fairness in network resource allocation," in *Proceedings of IEEE INFOCOM 2010*.

[23] C. Joe-Wong, S. Sen, T. Lan, and M. Chiang “Multiresource Allocation: FairnessâĂŞEfficiency Tradeoffs in a Unifying Framework,” *IEEE/ACM Transactions on Networking*, vol. 21, no. 6, pp. 1785–1798, Dec. 2013.

[24] W. Wang, B. Li, and B. Liang. “Dominant resource fairness in cloud computing systems with heterogeneous servers,” in *Proceedings of IEEE INFOCOM*, pp. 583–591, 2014.

[25] A. Ghodsi, V. Sekar, M. Zaharia, and I. Stoica, “Multi-resource fair queueing for packet processing,” in *Proceedings of ACM SIGCOMM 2012*.

[26] L. Popa, A. Krishnamurthy, S. Ratnasamy, and I. Stoica, “FairCloud: Sharing the network in cloud computing,” in *Proceedings of the 10th ACM Workshop on Hot Topics in Networks Article*, 2011.

[27] G. Lee, B.-G. Chun, and R. H. Katz, “Heterogeneity-aware resource allocation and scheduling in the cloud,” in *Proceedings of the 3rd USENIX Conference on Hot Topics in Cloud Computing*, 2011.

[28] Google Cluster Data. http://code.google.com/p/googleclusterdata/

[29] Google Research Blog. http://googleresearch.blogspot.ca/2011/11/more-google-cluster-data.html

[30] Q. Zhang, M. F. Zhani, R. Boutaba, and J. L. Hellerstein, "Dynamic heterogeneity-aware resource provisioning in the cloud," *IEEE Transactions on Cloud Computing*, vol. 2, no. 1, pp. 14–28, Jan.-Mar. 2014.

[31] O. A. Abdul-Rahman and K. Aida, "Towards understanding the usage behavior of Google cloud users: the mice and elephants phenomenon," in *Proceedings of IEEE 6th International Conference on Cloud Computing Technology and Science (CloudCom)*, pp. 272–277, 2014.

[32] Max-Min Fairness Wikipedia. http://en.wikipedia.org/wiki/Max-min_fairness

[33] B. Caprita, W. C. Chan, J. Nieh, C. Stein, and H. Zheng "Group ratio round-robin: O(1) proportional share scheduling for uniprocessor and multiprocessor systems," in *Proceedings of 2005 USENIX Annual Technical Conference*, pp. 337–352, 2005.

[34] I. Stoica, S. Shenker, and H. Zhang. "Core-stateless fair queueing: Achieving approximately fair bandwidth allocations in high speed networks," in *Proceedings of ACM SIGCOMM*, pp. 118–130, 1998.

[35] Bennett, Jon CR, and Hui Zhang. "WF^2Q: Worst-case fair weighted fair queueing," in *Proceedings of IEEE INFOCOM*, pp. 120–128, 1996.

[36] A. Demers, S. Keshav, and S. Shenker. "Analysis and simulation of a fair queueing algorithm," in *Proceedings of ACM SIGCOMM 1989*.

[37] M. Zaharia, D. Borthakur, J. S. Sarma, K. Elmeleegy, S. Shenker, and I. Stoica "Job scheduling for multi-user mapreduce clusters," EECS Department, University of California, Berkeley, Technical Report No. UCB/EECS-2009-55, April 30, 2009.

[38] E. Friedman, A. Ghodsi, and C.-A. Psomas, "Strategyproof allocation of discrete jobs on multiple machines," in *Proceedings of the 15th ACM Conference on Economics and Computation*, pp. 529–546, 2014.

[39] A. Ghodsi, M. Zaharia, S. Shenker, and I. Stoica, "Choosy: Max-min fair sharing for datacenter jobs with constraints," in *Proceedings of the 8th ACM European Conference on Computer Systems*, pp. 365–378, 2013.

[40] A. Ghodsi, V. Sekar, M. Zaharia, *et al*. "Multi-resource fair queueing for packet processing," *ACM SIGCOMM Computer Communication Review*, vol. 42, no. 4, pp. 1–12, 2012.

[41] A. A. Bhattacharya, D. Culler, E. Friedman, *et al*. "Hierarchical scheduling for diverse datacenter workloads," in *Proceedings of the 4th Annual Symposium on Cloud Computing*, pp.4, 2013.