QoS-guaranteed Resource Provisioning for Cloud-based MapReduce

A THESIS SUBMITTED TO

THE SCIENCE AND ENGINEERING FACULTY

OF QUEENSLAND UNIVERSITY OF TECHNOLOGY

IN FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

Xiaoyong Xu

Science and Engineering Faculty

Queensland University of Technology

2016

Copyright in Relation to This Thesis

Statement of Original Authorship

The work contained in this thesis has not been previously submitted to meet requirements for an award at this or any other higher education institution. To the best of my knowledge and belief, the thesis contains no material previously published or written by another person except where due reference is made.

Signature:

Date: 2/8/2016

Abstract

MapReduce, a popular programming model for big data processing, has been successfully implemented on various computing platforms such as cluster computing and grid computing. The recent advent of cloud computing provides a new platform for MapReduce computations. In cloud computing, MapReduce has been implemented as a service that can be delivered to users over the Internet. By making use of the unlimited elastic cloud resources (virtual machines, or VMs) and the pay-as-you-go business model, cloud-based MapReduce is more powerful and scalable than MapReduce implementations on other platforms. Cloud-based MapReduce faces a new issue, QoS-guaranteed resource provisioning, which aims to guarantee the Quality-of-Service (QoS) of MapReduce computations while minimizing their operational cost. This issue is very challenging in dynamic environments because a fixed resource allocation may become under-provisioning, leading to QoS violation, or over-provisioning, leading to unnecessary increases in operational costs.

This thesis makes four contributions through solving the QoS-guaranteed resource problem of cloud-based MapReduce. Firstly, this thesis derives a series of theoretical results summarized in theorems and corollaries. The theoretical results define the conditions for the amount and time for resource scaling-up to guarantee QoS, and also define the condition on the amount for resource scaling-down to guarantee QoS. The theoretical results, applied in the resource provisioning framework, help the framework decide when and how much resource needs to be provisioned. These theoretical results are demonstrated through case studies.

Secondly, the thesis derives a new MapReduce placement algorithm to solve the MRP problem. The new algorithm helps the resource provisioning framework decide how to place new MapReduce computations on VMs with minimum costs. It utilizes heterogeneous VMs to meet the resource requirements of cloud-based MapReduce computations, and effectively saves more cost than the traditional placement algorithms. As demonstrated in the experimental

results, the new algorithm saves more cost than current placement algorithms.

Thirdly, the thesis derives a new MapReduce consolidation algorithm to solve the MRC problem. The new algorithm helps the resource provisioning framework decide how to consolidate the remaining MapReduce computations with minimum costs. Experimental results show that the operational cost of cloud-based MapReduce is greatly reduced through using this new algorithm.

Finally, the thesis derives a novel resource provisioning framework for cloud-based MapReduce. Incorporating the outcomes from the first three perspectives, the framework identifies the changes in the computation environment that lead to QoS violation or resource waste, and handles the events by applying practical resource provisioning algorithms to adjust resource provisioning. Using that mechanism, the framework guarantees its QoS of cloud-based MapReduce while reducing the operational cost of cloud-based MapReduce as much as possible when the computation environment dynamically changes. Experiments demonstrate that the framework can support a QoS guarantee and that it has a better performance on saving operational costs than current popular resource provisioning frameworks.

Keywords

MapReduce, cloud computing, quality of service, resource provisioning

Acknowledgments

I would like to express my deep gratitude to my principal supervisor, Dr. Maolin Tang for his constant encouragement and guidance. He has walked me through all the stages of my Ph.D research. Without his patient instruction and insightful criticism, the completion of my Ph.D research would not have been possible.

I would also like to own my appreciation to my associate supervisor, Prof. Yu-chu Tian. He gives me valuable suggestions and numerous supports throughout the course of my Ph.D work.

Special thanks to Queensland University of Technology and China Scholarship Council (CSC) for their support in the form of a CSC Top-Up scholarship and a CSC Scholarship, respectively.

Finally, I would like to thank my family and friends for their concerns. They give me great encouragement when I experience difficulties during my Ph.D research.

Table of Contents

Nomenclature

Abbreviations

IaaS	Infrastructure as a Service
IFFD	Iterative First Fit Decreasing
MRP	MapReduce Placement
MRC	MapReduce Consolidation
QoS	Quality of Service
VM	Virtual Machines

Symbols		**Chapter**
c^M	Number of existing map workers	Ch5
	Number of map workers before consolidation	Ch6
c^R	Number of existing reduce workers	Ch5
	Number of reduce workers before consolidation	Ch6
C_m	Migration cost	Ch6
C_v	Cost of using VMs	Ch6
D	Hard deadline of a MapReduce computation	Ch4
J	a MapReduce computation	Ch7
m	Number of map tasks, number of VM types	Ch4,5
m_o	Number of the map tasks completed during the period T_o	Ch4
m_t	Number of the remaining map tasks at the time t	Ch4
m'	Number of types of existing VMs	Ch5

M_0	Number of pre-provisioned map workers	Ch4
M_+	Number of the map workers to be scaled up	Ch4
M_-	Number of the map workers to be scaled down	Ch4
M^{CPU}	CPU requirement of a map worker	Ch5,6
M^{Mem}	Memory requirement of a map worker	Ch5,6
M'^{CPU}	CPU requirement of an existing map worker	Ch5
M'^{Mem}	Memory requirement of an existing map worker	Ch5
$\mathbb{M}_+$	Variable to which R_+ is mapped	
	in the reverse resource space	Ch4
n	Number of new MapReduce computations	Ch5
	Number of remaining MapReduce computations	Ch6
n'	Number of existing MapReduce computations	Ch5
p	Price of a VM	Ch5,6
r	Number of reduce tasks	Ch4
r_o	Number of the reduce tasks completed during the period T_o	Ch4
r_t	Number of the remaining reduce tasks at the time t	Ch4
R_0	Number of pre-provisioned reduce workers	Ch4
R_+	Number of the reduce workers to be scaled up	Ch4
R_-	Number of the reduce workers to be scaled down	Ch4
R^{CPU}	CPU requirement of a reduce worker	Ch5,6
R^{Mem}	Memory requirement of reduce worker	Ch5,6
R'^{CPU}	CPU requirement of an existing reduce worker	Ch5
R'^{Mem}	Memory requirement of an existing reduce worker	Ch5
$\mathbb{R}_+$	Variable to which R_+ is mapped	
	in the reverse resource space	Ch4
t	Time of resource scaling	Ch4
t^A	Arrival time of a MapReduce computation	Ch7

t^M	Number of map workers needed to guarantee QoS	Ch5
t^R	Number of reduce workers needed to guarantee QoS	Ch5
T_E	End time of the whole system	Ch7
T_m^{avg}	Average duration of map tasks	Ch4
T_m^{max}	Maximum duration of map tasks	Ch4
$T_{m+}(M_+)$	Duration of map phase after resource scaling up	Ch4
$T^*_{m+}(\mathbb{M}_+)$	Function to which $T_{m+}(M_+)$ is mapped	
	in the reverse resource space	Ch4
$T_{m-}(M_-)$	Duration of map phase after resource scaling down	Ch4
$T^*_{m-}(\mathbb{M}_-)$	Function to which $T_{m-}(M_-)$ is mapped	
	in the reverse resource space	Ch4
T_o	Time delay of resource scaling-up	Ch4
	Basic charge time unit of a VM	Ch7
T_r^{avg}	Average duration of reduce tasks	Ch4
T_r^{max}	Maximum duration of reduce tasks	Ch4
$T_{r+}(R_+)$	Duration of reduce phase after resource scaling up	Ch4
$T^*_{r+}(\mathbb{R}_+)$	Function to which $T_{r+}(R_+)$ is mapped	
	in the reverse resource space	Ch4
$T_{r-}(R_-)$	Duration of reduce phase after resource scaling down	Ch4
$T^*_{r-}(\mathbb{R}_-)$	Function to which $T_{r-}(R_-)$ is mapped	
	in the reverse resource space	Ch4
$T_+(M_+, R_+)$	Total MapReduce duration after resource scaling up	Ch4
$T^*_+(\mathbb{M}_+, \mathbb{R}_+)$	Function to which $T_+(M_+, R_+)$ is mapped	
	in the reverse resource space	Ch4
$T_-(M_-, R_-)$	Total MapReduce duration after resource scaling down	Ch4
$T^*_-(\mathbb{M}_-, \mathbb{R}_-)$	Function to which $T_-(M_+, R_-)$ is mapped	
	in the reverse resource space	Ch4

T^E	End time of a worker	Ch7
T^S	Start time of a worker	Ch7
$\mathrm{T}_k(\mathbb{W})$	Function calculating the duration	
	of a MapReduce computation	Ch7
$\mathrm{T}_j(\mathbb{W})$	Function calculating the duration of a VM	Ch7
V	A new VM instance	Ch5,6,7
V^{CPU}	CPU capacity of a VM	Ch5,6,7
V^{Mem}	Memory capacity of a VM	Ch5,6,7
V^s	Worker placement of a new VM	Ch5
	Worker placement of a VM before consolidation	Ch6
$\mathbb{V}$	Multiset of new VMs	Ch5
	Multiset of existing VMs	Ch6
V'	An existing VM instance	Ch5
$V's$	Worker placement of an existing VM	Ch5
	Worker placement of a VM after consolidation	Ch6
$\mathbb{V}'$	Multiset of existing VMs	Ch5
W	a worker	Ch7
W^{CPU}	CPU requirement of a worker	Ch7
W^{Mem}	Memory capacity of a worker	Ch7
$\mathbb{W}$	Multiset of workers	Ch5,6,7
x^M	Number of new map workers	Ch5
	Number of map workers after consolidation	Ch6
x^R	Number of new reduce workers	Ch5
	Number of reduce workers after consolidation	Ch6

Greek Letters

α	Binary number	Ch4
β	Binary number	Ch4

γ	Binary number	Ch4

Superscripts

max	Maximum value	Ch4
min	Minimum value	Ch4
up	Upper bound	Ch4
low	Lower bound	Ch4

Subscripts

i	Integer representing the i^{th} MapReduce computations	Ch5,6
	Integer representing the i^{th} worker	Ch7
j	Integer representing the j^{th} type of VMs	Ch5,6
	Integer representing the j^{th} VM	Ch7
k	Integer representing the k^{th} VM	Ch5,6
	Integer representing the k^{th} MapReduce computation	Ch7

List of Figures

List of Tables

Chapter 1

Introduction

This chapter gives a brief background and outlines the motivation of this research. It then presents the research questions and the research problems. It concludes with the contributions of this research and a brief thesis outline.

1.1 Research Background

Big data analytics has recently become a major concern for modern enterprises. A large number of data sets are generated every day: for example, from Internet, public media and the daily activities of enterprises. These large data sets hide massive values, helping enterprises improve their productivity and competitiveness. A new challenge that is how to use big data to create values efficiently and effectively. To face that challenge, many major enterprises have started their big data analytics projects. For example, Microsoft has released the Analytic Platform System, which maintains and analyses big data sets, and has also provided the Azure Intelligent Systems Service, which is used to collect and manage the big data from Internet of Things [Numoto, 2015]. IBM has invested over 24 billions of dollars to build a Big Data and Analytics powerhouse during the last few years [Versace, 2014].

The enormous demands for big data analytics have boosted the development of parallel programming models. The most popular parallel programming model is MapReduce, which was first proposed by Google [Dean and Ghemawat, 2008]. When handling big data sets, MapReduce first breaks a large computation into a number of sub-tasks called *map tasks*, and processes these map tasks on a cluster of machines to generate a set of intermediate data in

parallel. It then assigns a number of sub-tasks called *reduce tasks* to the machines to process the intermediate data in parallel. Finally, it returns the required results. Many huge computations are able to be completed in very short durations using MapReduce. Due to its capability of large-scale data processing, MapReduce has been widely applied in many commercial and scientific applications, such as data mining [White et al., 2010, Wu et al., 2014], bioinformatics [Meng et al., 2011, Taylor, 2010] and machine learning [Chu et al., 2007, He et al., 2013].

An example of a MapReduce application for processing big files, called WordCount, is used to count the frequency of the 26 letters in the alphabet in a big file. The file is first partitioned into a number of file splits with the same size, and each split is an input of a map task. The map tasks are assigned to a cluster of machines. A map function is used to iterate each line of the file split, and counts the number of the letters in the line. After counting all letters in the file split, a list of key-value pairs, $List < letter, countNum >$, is generated, where $countNum$ is the count number of a letter. This list is then partitioned into several parts, according to the number of reduce tasks by a hash function $hash(Hashcode(letter)ModuloReduceID$, and each part of the list is an input of a reduce task. All the reduce tasks are transferred to the cluster of machines where all the key-value pairs are merged, sorted, and the list $List < letter, List < countNum >>$ are generated. For each reduce task, a reduce function is used to calculate the total number of $List < countNum >$ of each letter, and finally the list $List < letter, totalCountNum >$ is produced, where $totalCountNum$ is the total number of each of the letters counted in the big file.

The map and reduce tasks of a MapReduce computation are executed by two kinds of basic computing units, *map workers* and *reduce workers*, respectively. Map/reduce workers are placed on a cluster of machines so that they can acquire resources such as CPUs and memories from the machines to execute map/reduce tasks. The placement of map/reduce workers on machines identifies which machine executes the map/reduce tasks. A map/reduce worker for a MapReduce computation can be taken as a basic resource unit for that MapReduce computation, which abstracts the resource (i.e. CPU, memory) requirements of the map/reduce tasks of that MapReduce computation. The resource requirements of the MapReduce computation are indicated by the number of workers, rather than by the quantities of these CPUs or memories. A map/reduce worker also identifies the position where the map/reduce tasks are processed.

MapReduce is originally designed for cluster environments. In this sort of environment, end

users have to physically set up and maintain a cluster of machines for MapReduce computations. The cluster size can be very huge. In 2014, Yahoo! maintained a very large MapReduce cluster consisting of more than 100,000 CPUs in over 40,000 servers [Tavangar, 2014]. A very large monetary investment and a very big professional technical team are required to build and maintain such a big MapReduce cluster, which most enterprises can not afford. The cluster size is usually fixed and cannot be adjusted as the MapReduce workloads change. In particular, when the MapReduce workloads increase dramatically, the cluster hardly meets the performance requirements of the MapReduce computations. In contrast, when the MapReduce workloads decrease, a number of machines are idle, leading to resource waste. Thus, the cluster-based MapReduce is not cost-efficient, limiting its usage by small enterprises.

The emergence of cloud computing provides a new computation platform for MapReduce. Cloud computing provides the reliable and elastic cloud resource (Virtual Machines, or VMs) [Buyya et al., 2009] to MapReduce computations in a pay-as-you-go manner [Zhou and He, 2014]. VMs for the MapReduce computations are available at any time and are charged by usage durations. The number of VMs for MapReduce computations can be adjusted on demand. Multiple types of VMs can be used by MapReduce computations. Different types of VMs have different resource (i.e. CPUs, memories) capacities and prices. The VMs with larger resource capacity usually have the higher prices.

Compared with cluster-based MapReduce, cloud-based MapReduce is more powerful, scalable and cost-effective. As the MapReduce workloads increase, cloud-based MapReduce rents more VMs from clouds, or rents more powerful VMs to improve its computation capacity, so as to meet the performance requirements of its workloads. As the MapReduce workloads decrease, cloud-based MapReduce reduces the number of VMs or uses cheaper VMs to save the operational costs of cloud-based MapReduce. Thus, more MapReduce platforms are being moved from clusters to clouds. Amazon Elastic MapReduce [Amazon, 2015] and Azure MapReduce [Gunarathne et al., 2010] are two typical examples of cloud-based MapReduce.

1.2 Research Motivation

The emergence of cloud-based MapReduce raises two important issues. One issue is how to guarantee the Quality of Service (QoS) of cloud-based MapReduce. In cloud computing,

MapReduce can be delivered to end users as a service via Internet. The quality of the service, or QoS, should be guaranteed; this is negotiated by the MapReduce service provider and the end users. An essential QoS metric is a hard deadline, especially for the hard real-time systems in clouds, in which a deadline miss may result in a system failure or even a disaster [García-Valls et al., 2014]. The QoS metric particularly in this study is hard deadline. The other important issue is how to minimize the operational cost of the cloud-based MapReduce. To complete the MapReduce computations, the cloud-based MapReduce needs to rent the VMs from clouds and to pay for the usage of the VMs. Thus, the cost of using VMs constitutes the operational cost of the cloud-based MapReduce.

Then the new problem, *QoS-guaranteed resource provisioning*, emerges in cloud-based MapReduce. This new problem is how to guarantee the QoS (i.e. hard deadline) of the cloud-based MapReduce while minimizing its operational cost. There are two challenges for solving that problem. One challenge is to determine the amount of VM provisioning needed for the cloud-based MapReduce computations. As cloud computing is a dynamic environment, the VMs or network for cloud-based MapReduce computations may experience performance variability from time to time due to the shared underlying infrastructures [Schad et al., 2010]. The execution time of the same MapReduce computation could be different at different times. Also, the workloads in cloud computing usually vary over time, and the arrival times of new MapReduce computations are not known in advance [Ganapathi et al., 2010]. Therefore, the number of pre-provisioned VMs cannot be accurately estimated. Consequently, either the QoS will not be met if there are not enough pre-provisioned VMs, or unnecessary costs will be incurred if pre-provisioned VMs are over supplied.

Another challenge is selecting the right type and number of VMs for a MapReduce computation and the placement of the MapReduce computation on the selected VMs. The resource requirements of MapReduce computations are varied, and different types of MapReduce computation may consume different numbers and types of VMs. Also, different placements for the same MapReduce computation consume different types and numbers of VMs, incurring different running costs.

Currently, there has been some success in the investigation of the resource provisioning for MapReduce. Some have investigated how to guarantee the QoS of MapReduce under cluster environments [Hwang and Kim, 2012, Kc and Anyanwu, 2010, Polo et al., 2010, Shi

and Hong, 2013, Verma et al., 2011b, Zhang et al., 2014b]. Others have studied the resource provisioning for MapReduce in a cloud environment [AbdelBaky et al., 2012, Cardosa et al., 2011, Chen et al., 2014b, Herodotou et al., 2011, Lama and Zhou, 2012, Malekimajd et al., 2014, Mattess et al., 2013, Palanisamy et al., 2014, Rao and Reddy, 2012]; but their approaches do not always guarantee the QoS of MapReduce. Therefore, the QoS-guaranteed resource provisioning problem is still an open question in cloud-based MapReduce.

This research aims to solve the QoS-guaranteed resource provisioning problem in cloud-based MapReduces. Through this research presented by this thesis, the QoS of cloud-based MapReduce will be guaranteed and the operational cost of cloud-based MapReduce should be reduced as much as possible.

1.3 Research Questions

This research investigates the QoS-guaranteed resource provisioning for cloud-based MapReduce: this provisioning aims at guaranteeing the QoS (i.e. hard deadline) of cloud-based MapReduce while minimizing its operational cost. Cloud-based MapReduce is a computation platform in cloud computing environments. In cloud-based MapReduce, MapReduce computations continuously arrive, but the arrival times are not known in advance. Cloud-based MapReduce rents VMs from a public cloud to execute the MapReduce computations. When the computation environment changes, the resource provisioning for MapReduce computations might be insufficient, which leads to QoS violation, or could be more than sufficient, which incurs unnecessary operational costs. Cloud-based MapReduce supports the dynamical adjustment of resource provisioning as the computation environments vary. When the initial resource provisioning is insufficient, cloud-based MapReduce can scale up resource provisioning to speed up computation to meet the QoS. When the initial resource provisioning is more than sufficient, it can scale down resource provisioning to save operational costs. This explains the QoS-guaranteed resource provisioning for cloud-based MapReduce.

The QoS-guaranteed resource provisioning for cloud-based MapReduce involves several research questions. Firstly, cloud-based MapReduce needs to know when to implement the resource provision. When the new MapReduce computations continuously arrive, cloud-based MapReduce should decide the right timing of the initial resource provisioning for those new

MapReduce computations, such that the QoS cannot be violated because of intervening too late. Also, it should know when to scale up resource provisioning to avoid the QoS violation caused by insufficient resource provisioning, and when to scale down resource provisioning to avoid the cost increase incurred by resource over-provisioning. That raises the first research question:

Question 1: how can the timing of the resource provisioning for MapReduce computations be determined?

Having decided the timing of the resource provisioning, cloud-based MapReduce should calculate how much resource is required for the MapReduce computations. For new MapReduce computations, it has to determine the amount of the initial resource provisioning such that the QoS can be met while the operational cost is minimized. For MapReduce computations requiring resource scaling-up, it should decide the amount of resource to be added, such that the QoS is guaranteed while the operational cost is minimum. Meanwhile, for MapReduce computations requiring resource scaling-down, cloud-based MapReduce needs to know how much resource can be reduced at most, such that the operational cost is minimized while the QoS is still guaranteed. That raises the second research question:

Question 2: how can the amount of the resource provisioning for MapReduce computations be determined?

Once the amount of the resource provisioning is determined, cloud-based MapReduce needs to organize the right type and number of VMs to meet the resource requirement, and also needs to assign the MapReduce computations to the VMs. That process will be discussed in two situations. The first situation is when new MapReduce computations arrive or resource scaling-up needs to be done. In this situation, cloud-based MapReduce has to determine the size and type of new VMs and the placement of the MapReduce computations on the VMs, such that the cost of using VMs is minimized while the resource requirement is met. That raises the third research question:

Question 3: how can the number and types of VMs, and the placement of MapReduce computations on VMs be selected?

The second situation is when some MapReduce computations are completed. In this situation, some VMs could release redundant resources, so the placement of the MapReduce

computations needs to be re-organized to allow the idle VMs to be shut down to save operational cost. That raises the fourth research question:

Question 4: how can MapReduce computations be consolidated on VMs?

Answering all these research questions will solve the QoS-guaranteed resource provisioning problem in cloud-based MapReduce.

1.4 Research Problems

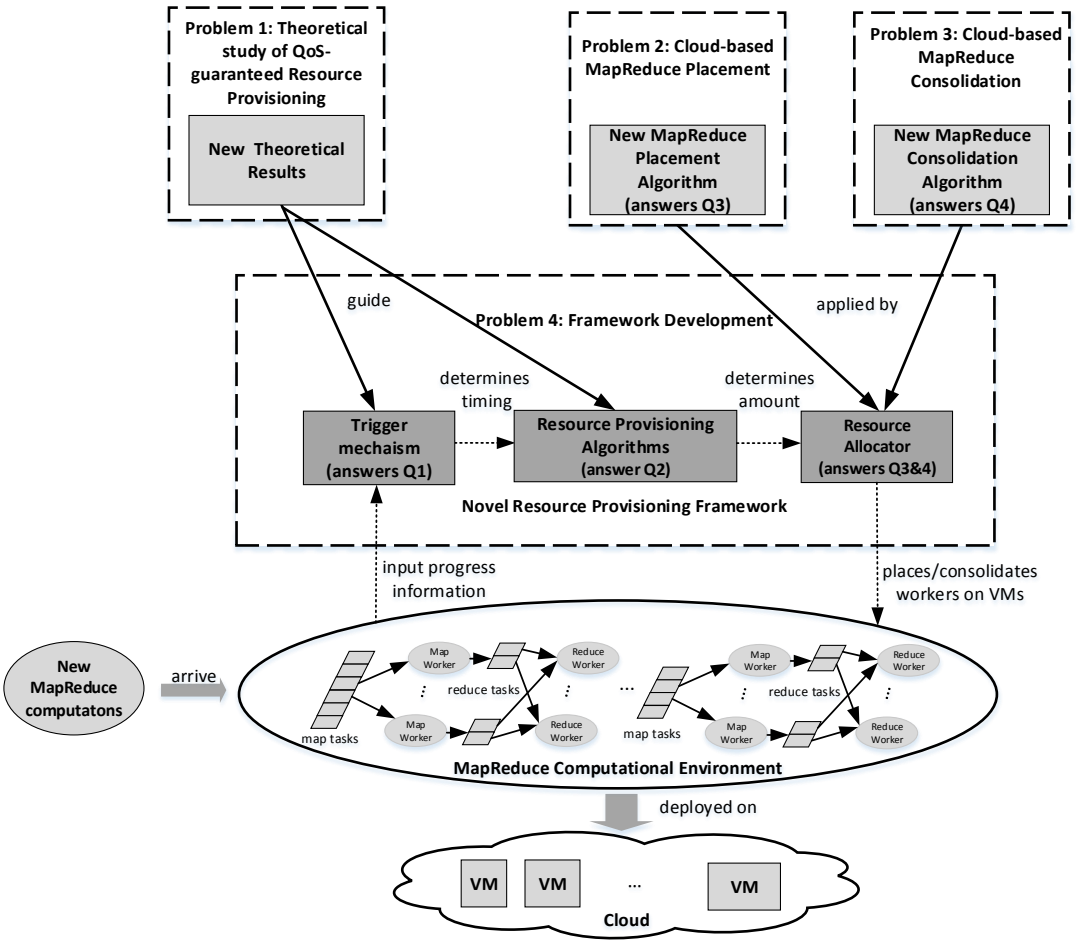


Figure 1.1: Relationship of the four research problems

This thesis answers these four research questions by proposing a new resource provisioning framework for cloud-based MapReduce. As shown in Figure 1.1, the framework consists of three parts, the trigger mechanism, the resource provisioning algorithms and the resource allocator. The trigger mechanism determines the timing of resource provisioning, which answers Question 1. The resource provisioning algorithms decide the amount of resource provisioning,

which answers Question 2. The resource allocator determines the VM provisioning and the placement of MapReduce computations on VMs, which answers Question 3. The resource allocator also conducts the consolidation of MapReduce computations on VMs, which answers Question 4.

To develop such a framework, four research problems need to be solved in advance. The first is the theoretical study of QoS-guaranteed resource provisioning, which lays the theoretical foundation for the development of the framework. The second and third problems are the cloud-based MapReduce placement and the cloud-based MapReduce consolidation, respectively. Study of these two problems helps the resource allocator of the framework guide determine the VM provisioning, the MapReduce placement and the MapReduce consolidation. The fourth one is the QoS-guaranteed resource provisioning. Based on the outcomes of the first three research problems, the study of the fourth research problem derives the complete resource provisioning framework, which finally solves the QoS-guaranteed resource provisioning problem in cloud-based MapReduce.

A more detailed description of these four research problems is given as follows.

1.4.1 Problem 1: Theoretical Study of QoS-guaranteed Resource Provisioning

The theoretical study of QoS-guaranteed resource provisioning seeks to find how to quantify the conditions for QoS-guaranteed resource provisioning on amount and timing. These theoretical results are used to guide the trigger mechanism of the framework to determine the timing of the resource provisioning. They are also used to help the resource provisioning algorithms of the framework to decide the amount of the resource provisioning.

The study of Problem 1 is broken into the studies of the three issues: the scaling-up issue, the scaling-down issue, and the latest intervention time issue.

The scaling-up issue is raised when new MapReduce computations arrive or when the initial resource provisioning is insufficient to meet the deadlines of the running MapReduce computations. For the new MapReduce computations, there is no resource pre-provisioned and it is necessary to scale up resources to start the execution of the new MapReduce computations. For the running MapReduce computations, if there is a performance degradation on the VMs or network, the computation progress will fall behind schedule. Without resource scaling-up, the

deadlines will be missed, leading to QoS violation. To avoid deadline misses, it is necessary to scale up resource to accelerate the computation progress. Then the scaling-up issue is raised: how to find the condition for resource scaling-up on an additional amount to guarantee QoS.

The latest intervention time issue characterizes the situation in which the resource scaling-up is activated too late to meet the deadlines. In this case, no pending tasks are left and consequently any additional resources do not help speed up the computation. To avoid this situation, the latest intervention time issue needs to be addressed. This issue is how to find the condition for resource scaling-up on intervention time to guarantee QoS.

The scaling-down issue describes the scenario when the pre-provisioned resource is over-provisioned for guaranteeing MapReduce deadlines. If there is a performance promotion on the VMs or network, the computation progresses ahead of the schedule. To minimize resource consumption, the resource provisioning needs to be scaled down. The scaling-down issue is then raised: how to find the condition for resource scaling-down on removal amount to guarantee QoS.

1.4.2 Problem 2: Cloud-based MapReduce Placement

The Cloud-based MapReduce placement problem is how to choose the types of VMs and the number of VMs of each type and then how to place the MapReduce computations on VMs. The problem aims at minimizing the cost of using VMs while meeting the resource requirement. Through the study of this problem, a cloud-based MapReduce placement algorithm is derived and applied by the resource allocator to answer Question 3. The algorithm helps the resource allocator of the framework to decide the VM provisioning and the placement of MapReduce computations on VMs.

1.4.3 Problem 3: Cloud-based MapReduce Consolidation

The Cloud-based MapReduce consolidation problem is how to re-assign the placement of the MapReduce computations on the VMs. The problem focuses on minimizing the operational cost while meeting the resource requirement. Through the study of this problem, a cloud-based MapReduce consolidation algorithm is derived and used by the resource allocator to answer Question 4. The algorithm guides the resource allocator of the framework to consolidate the

MapReduce computations on VMs.

1.4.4 Problem 4: Development of the QoS-guaranteed Resource Provisioning Framework

This problem is how to develop a new resource provisioning framework for cloud-based MapReduce. Through the study of this problem, a resource provisioning framework for cloud-based MapReduce that guarantees the QoS of cloud-based MapReduce while minimizing the operational cost in dynamical computational environments is derived. The framework answers Question 1 through introducing a trigger mechanism. This trigger mechanism is based on the theoretical results of the QoS-guaranteed resource provisioning timing conditions. It determines the timing of the initial resource provisioning for new MapReduce computations, and also the timing of the resource scaling-up/down for the MapReduce computations at runtime.

By introducing several resource provisioning algorithms, the framework answers Question 2. The resource provisioning algorithms apply the theoretical results of the QoS-guaranteed resource provisioning the amount conditions, and calculate the minimum required amount of the resource provisioning for the MapReduce computations to meet QoS.

Using a resource allocator the framework also answers Questions 3 and 4. The resource allocator applies the cloud-based MapReduce placement algorithm from the study of Problem 2 to conduct the VM provisioning and the MapReduce placement. That application answers Question 3. Meanwhile, the resource allocator adopts the cloud-based MapReduce consolidation algorithm from the study of Problem 3 to implement MapReduce consolidation. That application answers Question 4.

1.5 Research Assumptions and Scope

This section introduces several important assumptions on cloud-based MapReduce, that will be used in studying the four research problems. The research scope is identified in these assumptions.

The first assumption is that the cloud-based MapReduce studied in this research is a user of Infrastructure as a service (IaaS) provided by cloud computing. The cloud-based MapReduce

uses the VMs rented from IaaS to execute MapReduce computations, and the only operational cost of cloud-based MapReduce is the cost of using VMs. The details about the physical machines and networks in cloud data centres are transparent to the cloud-based MapReduce. It has no information about how the VMs are placed on physical machines, how they are migrated between physical machines, and how the data are transferred and stored in cloud data centres. Therefore, the cloud-based MapReduce in this study does not consider the costs incurred in the cloud data centres like the energy consumption and the migration cost of VMs.

The second assumption is that the cloud-based MapReduce studied in this research does not handle the configuration and the scheduling of the map/reduce tasks of a MapReduce computation. The cloud-based MapReduce tackles only the amount and timing of resource provisioning for MapReduce computations. The configuration and the scheduling of the map/reduce tasks of a MapReduce computation are out of the scope of this research, and are customized by the end users of the cloud-based MapReduce. Before submitting a MapReduce computation, the end users need to configure the number of the map/reduce tasks, the split size of every map/reduce input, the data placement and the deadline of the MapReduce computation. The end users also need to configure the scheduling policy of the map/reduce tasks, which determines the execution sequence of the map/reduce tasks. When the MapReduce computation is submitted, the cloud-based MapReduce executes it using the end users' the configuration and scheduling policy.

The third assumption is that the cloud-based MapReduce studied in this research considers two basic types of resources for MapReduce computations: i,e. CPUs and memories. Other resources such as I/O and disk spaces are not considered. The cloud-based MapReduce is an end user of IaaS, but current IaaS does not provide the information about network I/O rates and disk I/O rates, so this information is not considered by the cloud-based MapReduce studied in this research. As MapReduce computations are usually not disk-bound, so the disk spaces are also not considered by the cloud-based MapReduce.

1.6 Major Contributions

Addressing the four research problems provides the four contributions outlined in this section:

- New theoretical results of QoS-guaranteed resource provisioning for cloud-based MapReduce. This research proposes a series of theoretical results for QoS-guaranteed resource provisioning in cloud-based MapReduce. These theoretical results characterize the required conditions of the amount and time of resource scaling for guaranteeing the QoS of cloud-based MapReduce.

- A new MapReduce placement algorithm for cloud-based MapReduce. The new placement algorithm helps the framework place new MapReduce computations on VMs with minimum operational costs.

- A new MapReduce consolidation algorithm for cloud-based MapReduce. The new consolidation algorithm helps the framework consolidate MapReduce computations on VMs with minimum operational costs when some MapReduce computations are completed.

- A novel QoS-guaranteed resource provisioning framework for cloud-based MapReduce. This framework guarantees the QoS of cloud-based MapReduce with minimum operational costs by dynamically adjusting the resource provisioning when the environment changes.

1.7 Thesis Outline

This thesis is organised in seven chapters:

- Chapter 2 reviews the literature related to this research topic. This chapter also identifies the uniqueness of this research and the research gaps that exist in the current literature.

- Chapter 3 presents the theoretical analysis of QoS-guaranteed resource provisioning. The theoretical results derived from this chapter are used in the design of the new resource provisioning framework in Chapter 6, and lay the theoretical foundation for QoS-guaranteed resource provisioning.

- Chapter 4 proposes a new MapReduce placement algorithm that solves the cloud-based MapReduce consolidation problem.

- Chapter 5 proposes a new MapReduce consolidation algorithm that solves the cloud-based MapReduce consolidation problem.

- Chapter 6 develops a novel event-driven resource provisioning framework for cloud-based MapReduce. The framework adopts the results from Chapters 3, 4 and 5, and successfully solves the QoS-guaranteed resource provisioning problem.
- Chapter 7 summarizes the research presented in this thesis. Meanwhile, some future directions are given in the chapter.

1.8 List of Publications

From the research of this PhD project, seven papers have been published in or submitted to journals and conferences. The main results of Chapter 3 have been discussed in the following paper published in an IEEE transactions journal:

- Xu, X, Tang, M and Tian, Y.C. (2016). Theoretical results of QoS-guaranteed resource scaling for cloud-based MapReduce. *IEEE Transactions on Cloud Computing*, accepted on 8th, February, 2016, in press.

The main outcomes of Chapter 4 have been published in the following three papers:

- Xu, X and Tang, M (2015). A new approach to the cloud-based heterogeneous MapReduce placement problem. *IEEE Transactions on Services Computing*, accepted on 15th, May, 2015, in press.
- Xu, X and Tang, M (2014). A new grouping genetic algorithm for the MapReduce placement problem in cloud computing. In *Proceedings of 2014 IEEE Congress on Evolutionary Computation*, pages 1601-1608. (Tier 'A')
- Xu, X and Tang, M (2014). A more efficient and effective heuristic algorithm for the MapReduce placement problem in cloud computing. In *Proceedings of 2014 IEEE International Conference on Cloud Computing*, pages 264-271. (Tier 'B', acceptance rate: 18%)

Besides these, a paper which discusses the operation model of MapReduce has been published:

- Xu, X and Tang, M (2013). A comparative study of the semi-elastic and fully-elastic MapReduce models. In *Proceedings of the 2013 IEEE International Conference on*

Granular Computing (GrC), IEEE, Beijing Institute of Technology, Beijing, China, pages 380-385.

Chapter 2

Literature Review

This chapter reviews the published literature related to the research issue being investigated in this study. Section 2.1 reviews the literature regarding the state of the art of QoS-guaranteed resource provisioning for cloud-based MapReduce. Sections 2.2 to 2.5 review the publications related to the study's four research problems: the theoretical study of QoS-guaranteed resource provisioning, the cloud-based MapReduce placement, the cloud-based MapReduce consolidation and the resource provisioning frameworks for cloud-based MapReduce, respectively. Section 2.6 gives the chapter summary.

2.1 State of the Art of QoS-guaranteed Resource Provisioning for Cloud-based MapReduce

Guaranteeing the cloud-based MapReduce service quality, known as QoS-guaranteed resource provisioning, is an important issue. This issue also exists for non-MapReduce computations in cloud computing and for non-cloud-based MapReduce computations.

This section reviews the studies of QoS-guaranteed resource provisioning for both non-MapReduce computations in cloud computing (Section 2.1.1) and non-cloud-based MapReduce computations (Section 2.1.2), before reviewing the studies for cloud-based MapReduce computations (Section 2.1.3). In this section, those studies are each compared with our study to identify the differences between them and ours.

2.1.1 QoS-guaranteed Resource Provisioning for Non-MapReduce Computations in Cloud Computing

The review considers the research projects that focused on aspects of QoS-guaranteed resource provisioning for non-MapReduce computations in cloud computing, before noting their relevance for our study. Garg et al. [2011] studied the resource provisioning problem for different types of application workloads in cloud computing. They proposed a resource provisioning mechanism based on admission control, which would meet the QoS (i.e. deadline) requirements of those application workloads in cloud computing while maximizing the profit of the cloud service provider. However, as the workflows of their application workloads are different from that of the cloud-based MapReduce computations in this study, their technology cannot be applied in the QoS-guaranteed resource provisioning issue in this study.

Ai et al. [2011] studied the resource allocation and scheduling problems of multiple composite web services in cloud computing. They tried to minimize the running costs of multiple composite web services while meeting the quality of the services (i.e. deadlines). They solved the problems by using a Cooperative Coevolutionary Genetic Algorithm (CCGA). However, the structure and process of the multiple composite web services are totally different from those of MapReduce computations in this study, so their algorithm cannot be used for the QoS-guaranteed resource provisioning issue in this study.

Singh and Chana [2015] considered how to meet the required QoS level of cloud-based workloads while minimizing the usage cost of cloud resource. They identified multiple QoS metrics, such as execution time, resource utilization, and availability, for different types of cloud-based workloads. The QoS-based resource provisioning approach they proposed for their problem was efficient in reducing the usage cost of cloud resource while meeting multiple QoS metrics. However, the structure and process of the workloads in their research are different from those of the cloud-based MapReduce computations in this study. Their resource provisioning approach is used specifically for their workloads, rather than for cloud-based MapReduce computations.

Unlike those research examples, the research of Beloglazov et al. [2012] investigated how to meet the QoS (i.e. deadline) requirements of cloud-based applications such as workloads while minimizing the energy consumption of cloud data centers. They considered the variations in workloads, and proposed energy-aware resource allocation heuristics to solve that problem.

However, experimental results showed that the deadlines of the cloud-based workloads were not always met. In addition, the structure and process of the cloud-based workloads are different from those of cloud-based MapReduce computations. Thus, their heuristics cannot be applied to address the QoS-guaranteed resource provisioning issue in our research.

Xiao et al. [2013] explored how to satisfy the QoS (i.e. deadline) requirements of cloud-based applications with the minimum energy consumption of cloud data centers via dynamic resource allocation. They developed a set of heuristics which combined different types of cloud-based applications on VMs, so as to reduce the number of VMs used. However, our QoS-guaranteed resource provisioning issue considers the cost of using VMs, not the energy consumption. As the cloud-based applications studied in their work are very general, their resource provisioning approach is not specially designed for our cloud-based MapReduce computations. Thus, their approach cannot be used for the QoS-guaranteed resource provisioning issue in our research.

Zhang et al. [2014a] presented a heterogeneity-aware dynamic capacity provisioning approach for cloud data centers that aimed at meeting the QoS (i.e. deadline) requirements of cloud workloads while minimizing the energy consumption of cloud data centers. This approach dynamically adjusted the number of VMs to minimize the energy consumption and the scheduling delay. However, their approach cannot be used for our QoS-guaranteed resource provisioning issue since our issue is how to minimize the cost of using VMs rather than about energy consumption.

Daryani and Thakare [2015] proposed a scheduling algorithm for the cloud resource provisioning of cloud service providers. That algorithm aimed at adopting dynamic pricing strategies to maximize the revenues of cloud service providers while meeting the deadlines of cloud-based jobs. However, the cloud-based jobs studied in their work are very general. This scheduling algorithm is not designed specially for MapReduce computations, so cannot be directly used for MapReduce computations.

In researching the application of energy-efficient and QoS management in cloud computing, Sampaio and Barbosa [2016] analyzed the performance of three scheduling algorithms on energy efficiency. However, these scheduling algorithms are specially designed for minimizing energy consumption, rather than for the QoS-guaranteed resource provisioning for cloud-based MapReduce computations, which aims to minimize the cost of using VMs.

Several researchers adopted evolutionary approaches to address the deadline-constrained resource scheduling problems in cloud computing. Chen et al. [2015] proposed a genetic algorithm to solve a cost-minimization and deadline-constrained workflow scheduling mode in cloud computing. Their proposed algorithm focused on how to minimize the execution cost while meeting deadline constraints. Similarly, Li et al. [2015] studied the cloud resource scheduling problem which was how to minimize the overall workflow execution cost while satisfying deadline constraints. They designed a coevolutionary multiswarm particle swarm optimization (CMPSO) algorithm to solve that problem. Furthermore, Zhan et al. [2015] presented a comprehensive survey of evolutionary approaches for cloud resource scheduling problems. But all these evolutionary approaches are not specially designed for the scheduling problems of cloud-based MapReduce computations, and cannot be directly adopted to solve the QoS-guaranteed resource provisioning problem for cloud-based MapReduce computations.

In summary, there has been a variety of research on the QoS-guaranteed resource provisioning for non-MapReduce computations in cloud computing. However, as the resource provisioning approaches reviewed are not specially designed for cloud-based MapReduce computations, they cannot be directly used to address the QoS-based resource provisioning issue for the cloud-based MapReduce computations that is central to our study.

2.1.2 QoS-guaranteed Resource Provisioning for Non-cloud-based MapReduce

When considering QoS-guaranteed (i.e. deadline-guaranteed) resource provisioning for non-cloud-based MapReduce, the review focused first on five related research projects. Verma et al. [2008] proposed a resource provisioning mechanism to meet the deadlines of cluster-based MapReduce computations. Kc and Anyanwu [2010] proposed a realtime cluster scheduling approach to meet the user specified deadline constraints for MapReduce computations.

Polo et al. [2010] proposed a deadline-aware task scheduler for cluster-based MapReduce computations. This scheduler dynamically predicted the performance of the running cluster-based MapReduce computations and the scaled up/down resource provisioning for the cluster-based MapReduce computations. It could reduce the energy consumption of the cluster while meeting the deadline constraints of the cluster-based MapReduce computations.

Dong et al. [2011] developed a two-level MapReduce scheduler to address the deadline-guarantee issue for mixed realtime and non-realtime MapReduce computations in clusters. The

scheduler used a sampling-based approach to estimate the completion times of map/reduce tasks and then used an allocation model to dynamically adjust the resource provisioning for map/reduce task execution to maximize the cluster utilization while meeting the deadlines of the realtime and non-realtime MapReduce computation.

Liu et al. [2012] formulated the deadline-guaranteed resource provisioning problem in cluster-based MapReduce computations as a preemptive scheduling problem and developed a preemptive scheduling approach to address that scheduling problem.

All these research examples assume that the total number of machines for MapReduce computations is fixed. When the resource requirements of the workloads exceed the computing capacity of the cluster for the MapReduce computations, the deadlines of the MapReduce computations will be violated. Our study differs, in that it assumes that the machines for MapReduce computations can be added/removed on demand, so we are unable to use their mechanism in our research.

Four research projects were reviewed in the second group. Polo et al. [2013] studied the problem of meeting the deadline constraints of cluster-based MapReduce computations using a heterogeneous cluster environment. They proposed a scheduler to manage the MapReduce workloads to meet the high level of the deadline constraints. Wang et al. [2014] proposed a sequence-based scheduler to address the deadline-guarantee issue in cluster-based MapReduce computations. This scheduler optimized the sequence of the cluster-based MapReduce computations to reduce the deadline violation rate. Teng et al. [2015] proposed a maximum urgency scheduler for Hadoop MapReduce. This scheduler was applied in heterogeneous MapReduce environments with a low computation complexity and maximized the number of the MapReduce computations meeting their deadlines. All four schedulers presented by these different researchers cannot ensure that every MapReduce computation meets its deadline. As our study is how to meet the deadline of every MapReduce computation these schedulers cannot be used for this study.

The review then grouped three studies. Zhang et al. [2012] considered how to meet the deadline constraints of cluster-based MapReduce computations through optimizing data placement. They formulated the data placement optimization problem as an integer bi-level programming

problem and proposed a bi-level genetic algorithm to solve that programming problem. Similarly, Tang et al. [2012] proposed a task scheduler for meeting the deadline constraints of cluster-based MapReduce computations. This scheduler distributed data in terms of the capacity level of machines in a cluster, which improved data locality. Also, this scheduler estimated the task completion time according to the capacity level of machines in the cluster and then decided the resource provisioning for map/reduce task execution. Wang and Li [2015] studied the task scheduler for MapReduce computations in geo-distributed data centers on heterogeneous networks. Their work considered data locality and data transfer when designing the scheduler. However, the cloud-based MapReduce in our study is a computation platform deployed on top of cloud computing and cannot manage the data placement as the storage layer is transparent to it. Thus, these approaches cannot be used to address the QoS-guaranteed resource provisioning for cloud-based MapReduceconsidered in our study.

Further on schedulers, Li et al. [2014a] developed a deadline-enabled delay scheduler to address the deadline-guarantee issue in cluster-based MapReduce computations. This scheduler considered the issue of resource availability and optimized the delay decisions of resource provisioning for cluster-based MapReduce according to realtime resource availability and resource competition. But the resource provisioned to cloud-based MapReduce computations is ensured by Infrastructure as a Service (IaaS) with high-level availability, and the issue of resource availability does not exist in the cloud-based MapReduce of our study.

All the research projects (in Section 2.1.2) have considered the QoS-guarantee resource provisioning issue for cluster-based MapReduce computations. Other research has studied the resource provisioning for MapReduce computations in grid computing. Tang et al. [2010] presented a MapReduce framework, deployed on Desktop Grid, that had an ability of massive fault tolerance and high availability. This MapReduce framework also achieved linear speed-up on the classical MapReduce computations through a scalability test. He et al. [2012] developed a Hadoop MapReduce framework (HOG) executed on grid computing. Differing from the cluster-based MapReduce which utilized dedicated resources, HOG utilized free and opportunistic resources of the grid to execute MapReduce computations. This research project focused on improving the fault tolerance of HOG on three levels the node level, the rack level and the site level via managing resource provisioning.

Unlike other computing environments such as cluster computing and cloud computing,

the machines in opportunistic or volunteer environments are volatile and can be shut down at any time. Thus, enhancing availability is an important issue for MapReduce computations in these particular environments. Lin et al. [2010a] developed a MapReduce implementation, namely MOON, in opportunistic environments. MOON, extended from Hadoop MapReduce, utilized adaptive task- and data-scheduling algorithms to improve its availability. Lee and Figueiredo [2012], who studied how to provide MapReduce services with high availability in volunteer environments, adopted an uptime-based resource availability prediction method to improve the availability of the MapReduce computations using the volatile resources in volunteer environments. Kijsipongse and U-ruekolan [2014] deployed Hadoop MapReduce in a virtual volunteer computing environment. In that environment, the machines for MapReduce computations comprised a fixed number of dedicated computers with high availability and a variable number of volunteer computers with low availability. The MapReduce computations under that framework performed better on execution time, compared with the MapReduce computations using the dedicated computers.

However, the computational environment for the cloud-based MapReduce in this study is quite different from these three types of computing environments (i.e. grid computing, opportunistic computing and volunteer computing). The computing resource (i.e. VMs) are dedicated for the cloud-based MapReduce computations and the qualities of computing resource (i.e. availability and reliability) are ensured by cloud computing. Thus, the issues of availability and fault tolerance in these three types of computing environments do not exist in the cloud-based MapReduce.

In summary, the computational environments for non-cloud-based MapReduce computations are different from those for cloud-based MapReduce computations. The resource provisioning approaches used for non-cloud-based MapReduce computations are not suitable for addressing QoS-guaranteed resource provisioning for cloud-based MapReduce computations.

2.1.3 QoS-guaranteed Resource Provisioning for Cloud-based MapReduce

The QoS-guaranteed resource provisioning problems in cloud-based MapReduce are categorized into two types: *static resource provisioning problems* and *dynamic resource provisioning problems*.

Static Resource Provisioning Problems

The static resource provisioning problems relate to determining the amount of resource provisioning at a time point in order to guarantee the QoS (i.e. deadline) of cloud-based MapReduce with minimum operational cost. These problems usually assume that the computational environment for cloud-based MapReduce computations is stable, and that resource scaling is not required at runtime.

Several researchers have studied the static resource provisioning problems in cloud-based MapReduce. Herodotou and Babu [2011] investigated how to optimize the cluster size for MapReduce computations with deadline constraints. They modeled the impact of various MapReduce parameters, such as input size, number of map/reduce tasks and number of map/reduce workers, on the MapReduce computation time. They then adopted the model to determine the optimum cluster size for meeting the deadlines of cloud-based MapReduce computations. But they did not study how to optimize the cluster size for a running cloud-based MapReduce computation.

Similarly, Hwang and Kim [2012] aimed to deal with the resource provisioning problem for MapReduce in the cloud by minimizing the monetary cost of VMs while meeting the deadline constraints. In their interpretation of the problem, the computational environment was assumed to be stable, and the resource provisioning could not be changed once it was determined.

Lama and Zhou [2012] investigated the allocation of heterogeneous resources to cloud-based MapReduce computations for guaranteeing the deadlines with minimum operational costs. Their resource allocation problem did not consider how to change resource allocation when the cloud-based MapReduce computations were running.

Cardosa et al. [2012] studied how to determine the minimum number of VMs for cloud-based MapReduce computations while meeting the deadlines. Once the minimum number was determined, however, it could not be changed according to MapReduce computation progress.

Hwang and Kim [2012] investigated how to determine the number of VMs that cloud-based MapReduce computations needed for meeting the deadlines with the minimum of VM cost. The number of VMs was determined before the cloud-based MapReduce computations started running, and could not be changed at runtime.

Palanisamy et al. [2014] studied the resource provisioning problem that aimed at meeting the

deadlines of cloud-based MapReduce while minimizing the cost of using VMs. Their problem assumed that the computational environment did not change.

Chen et al. [2014b] sought to optimize the resource provision for cloud-based MapReduce computations to minimize the cost of using VMs while meeting the deadlines. Through solving this problem, they determined the amount of resource provisioning required before cloud-based MapReduce computations started running. But they did study how to change the amount of resource provisioning at runtime when the computational environment varied.

Malekimajd et al. [2014] worked on how to find the optimum cloud capacity to minimize the resource cost and job rejection penalties while meeting the dealings of cloud-based MapReduce computations. They formulated the problem into a linear programming problem and then proposed an algorithm to solve this problem. Once the optimum cloud capacity was derived, however, that capacity could not be changed at runtime.

Zhang et al. [2015] focused on the decision problem about how to determine the number and types of VMs for the cloud-based MapReduce computations with the deadline constraints. To solve that problem, they conducted a fast simulation-based framework for determining the right size and type of VMs to meet deadlines before the cloud-based MapReduce computations started running. However, they did not study how to add/remove VMs at runtime.

Thus, none of these static resource provisioning problems considered the changes in computational environments. None of them studied how to determine the amount of resource scaling at runtime. The QoS-guaranteed resource provisioning problem in this study differs from those problems in that it takes the changes in computational environments into account, and also investigates the determination of the amount of adding/removing resource at runtime.

Dynamic Resource Provisioning Problems

The dynamic resource provisioning problems refer to determining the amount of resource provisioning at runtime for guaranteeing the QoS (i.e. deadline) of cloud-based MapReduce with minimum operational cost. These problems usually assume that the computational environment for cloud-based MapReduce computations changes dynamically, and that resource scaling is required at runtime.

The following research has considered these dynamic resource provisioning problems in

cloud-based MapReduce. Byun et al. [2011] sought to determine the right amount of cloud resource for cloud-based MapReduce computations in order to minimize the usage cost of cloud resource while meeting the deadline requirements. They considered the dynamic changes in the computational environment, and proposed a heuristic to dynamically provision cloud resource to cloud-based MapReduce computations. However, their experimental results showed that their heuristic could not meet the deadline of every cloud-based MapReduce computation.

Wang et al. [2013] worked with the problem of how to schedule maps and/or reduce tasks to guarantee the deadlines of cloud-based MapReduce computations. In their problem, they studied how to adjust resource allocation at runtime, but the deadline constraints in their problem were soft ones.

Xiang et al. [2013] tried to solve the resource allocation problem for cloud-based MapReduce computations, which meant optimizing the Nash Bargaining Solutions with respect to deadlines and computation priorities. In their research problem, deadlines were optimization objectives, rather than constraints, which meant that deadline violation was possible.

Chen et al. [2014a] investigated how to manage the resource provisioning for cloud-based MapReduce computations when the computational environment changed. These researchers used Bipartite Graph modeling to develop a new MapReduce scheduler. This new scheduler could adjust resource provisioning when the computational environment changed and could improve data locality to reduce the deadline violation rate. But this scheduler could not ensure that the deadline of every cloud-based MapReduce computation was met.

Teng et al. [2014] considered the real-time scheduling problem for the cloud-based MapReduce computations. The objective of the problem was to improve the probability of meeting deadlines via dynamic resource provisioning, rather than by meeting hard deadlines.

Therefore, none of these dynamic resource provisioning problems studied how to meet the hard deadlines of cloud-based MapReduce computations. Deadline violation was allowed in their problems. On the contrary, the QoS-guaranteed resource provisioning problem in this study aimed at meeting hard deadlines: that is, meeting deadlines with 100 percent.

Alrokayan et al. [2014] found out how to dynamically provision resource to the cloud-based MapReduce computations under deadline constraints. They proposed a deadline-aware resource provisioning approach which could adjust resource provisioning using the information of data

sources and network throughput. However, because that information is unknown in the QoS-guaranteed resource provisioning problem central to our study, we cannot adopt their approach solve our problem.

Gandhi et al. [2015] focused on the resource auto-scaling problem for the MapReduce computations in clouds. They established a performance model for the relationship between MapReduce computation time and the system parameters such as input size and resource allocation, and then optimized the configurations of the system parameters to minimize the resource cost while meeting the deadlines. However, system parameters such as input size are not considered in our QoS-guaranteed resource provisioning problem, since they are configured by the MapReduce users.

Cheng et al. [2015] studied the deadline-aware scheduling problem for cloud-based MapReduce computations that would reduce deadline misses. This scheduling problem considered resource availability. A prediction model was used to estimate future resource availability to enable accurate estimation of MapReduce completion time. However, resource availability is not an issue in the QoS-guaranteed resource provisioning problem of this current research, since these issues can be addressed by scaling up cloud resource provisioning.

In summary, the QoS-guaranteed resource provisioning problems found in the current literature are different from the problem faced in this research. Those other research problems do not consider the changes in computational environments, are not aimed at meeting hard deadlines or are studied under different MapReduce environments. But the problem in this research considers the changes in cloud environments and its objective is to meet hard deadlines.

2.2 Theoretical Study of QoS-guaranteed Resource Provisioning

The literature on the first research problem, the theoretical study of the QoS-guaranteed resource provisioning, is reviewed in the first section (2.2.1). This theoretical study considers how to quantify the amount and timing of resource provisioning. The studies are categorized into two main groups in terms of their resource provisioning mechanism and their computing environments: the theoretical studies for non-cloud-based MapReduce and the theoretical studies for cloud-based MapReduce.

2.2.1 Theoretical Study for Non-cloud-based MapReduce

Research on resource provisioning for MapReduce in non-cloud computing is well represented. Hwang and Kim [2012], Verma et al. [2011b] considered how to determine the minimal amount of resource to meet the deadlines of cluster-based MapReduce computations before these computations start running. However, no research was found on how to quantify the amount of resource provisioning at the runtime of the MapReduce computations.

In comparison, other authors considered workload variations, applying various resource scaling techniques to meet the deadlines of MapReduce computations: for example, Kc and Anyanwu [2010], Li et al. [2014a,b], Polo et al. [2011], Shi and Hong [2013], Zhang et al. [2014b]. When the workloads increase, the resource is usually scaled down for those computations with low priority, and scaled up for the computations with high priority. When doing so, the total resource capacity of the cluster remains unchanged.

All that research investigated the resource provisioning for MapReduce computations in non-cloud environments. As our study investigates the resource provisioning for MapReduce computations in cloud environments, our research differs from the topics presented in the work of those researchers.

2.2.2 Theoretical Study for Cloud-based MapReduce

In cloud computing, there is some research on the amount of initial resource provisioning needed to meet the QoS of MapReduce computations. Researchers have derived the theoretical results by solving an Integer Linear Programming problem. To study the amount of resource provisioning for ad-hoc MapReduce computations in clouds, Lama and Zhou [2012] introduced a two-phase machine learning and optimization framework to optimize the operational cost while achieving the QoS goals. Chen et al. [2014b] searched for the optimal amount of map and reduce slots to execute the cloud-based MapReduce computations with minimum costs.

Palanisamy et al. [2014] proposed a resource provisioning framework for cloud-based MapReducethat would minimize the monetary cost of using cloud resources while meeting the deadlines of MapReduec computations. Herodotou et al. [2011] developed a system named Elastisizer to answer the cluster sizing problems for the cloud-based MapReduce computations.

However, all these theoretical results are used to determine the amount of the initial resource

provisioning for new MapReduce computations, rather than the amount of resource scaling at runtime. In our research, not only the amount of initial resource provisioning, but also the amount of resource scaling at runtime, is required to be determined; thus their theoretical results cannot be applied in our research.

Unlike the research just described, other research has investigated the amount of runtime resource provisioning needed for cloud-based MapReduce. AbdelBaky et al. [2012] gave a rough estimation for the number of VMs required to meet the deadlines of the MapReduce computations in CometCloud. Mattess et al. [2013] studied dynamic resource provisioning to guarantee soft deadlines for cloud-based MapReduce. Malekimajd et al. [2015] explored the upper/lower bounds for MapReduce computation time in cloud systems, and then calculated the minimum amount of resource for MapReduce computations to meet their soft deadlines. Ruiz-Alvarez et al. [2015] calculated the optimum amount of resource allocation for the cloud-based MapReduce computations with deadline constraints.

The theoretical results from these four research projects were used for computations with soft deadlines, not with hard deadlines. Unlike that research focus on soft deadlines, our study presents a theory about how to guarantee the hard deadlines of MapReduce computations.

Although Cardosa et al. [2011] studied the amount of resource provisioning needed for the map phases of MapReduce computations, their theoretical results did not really guarantee QoS of MapReduce computations, especially when there was a performance degradation in the reduce phases of the MapReduce computations.

In addition, almost all the current research is about how to determine the amount of resource provisioning, but it ignores how to determine the timing of resource provisioning. The timing issue is essential to QoS-guaranteed resource provisioning, since resource provisioning that is too late could lead to missed deadlines. Therefore, our study considers the timing of resource provisioning for cloud-based MapReduce computations.

In summary, the theoretical results of amount and timing of resource provisioning found in current literature do not always guarantee the QoS of cloud-based MapReduce. There is not yet a detailed theoretical analysis of the QoS-guaranteed resource provisioning. Our study establishes theoretically the resource amount for QoS-guaranteed resource provisioning, and derived mathematically the latest intervention time for avoiding QoS violation. These theoretical results can be used to guide the QoS-guaranteed resource provisioning for cloud-based MapReduce.

2.3 MapReduce Placement

The research on the second research problem, the MapReduce placement (MRP) problem, is reviewed in this section. The studies are categorized into two groups in terms of computing environments: non-cloud-based MapReduce consolidation and cloud-based MapReduce consolidation. The review identifies the gaps in these two groups of studies and illustrates the innovation of our study.

2.3.1 Non-cloud-based MapReduce Placement

This review considers the research projects related to the MapReduce placement problem in non-cloud-based MapReduce computations. Zaharia et al. [2008] followed the default Hadoop configuration for the worker placement on computing nodes [White, 2009], developing a scheduler called LATE for Hadoop MapReduce, deployed on heterogeneous environments to reduce job execution time. Lin et al. [2010b] studied the adaptive task- and data-scheduling algorithms in MOON, a MapReduce implementation under the volunteer computing environment, and improved the MapReduce performance on execution time under that environment with the volatility of resources and a high rate of node unavailability. They also did not consider how to assign slots to slave nodes; they just implemented the default Hadoop settings. Wolf et al. [2010] proposed a slot-allocation scheduling optimizer to provide a minimum number of slots to MapReduce workloads. This optimizer aimed at optimizing some metrics, such as execution time, while ensuring the same minimum and maximum slot guarantees as in HFS. In their work, the slot assignment on computing nodes followed a default configuration. Kc and Anyanwu [2010] developed a constraint-based Hadoop scheduler, based on a job execution cost model, to meet the deadline constraints specified by users. They also used the default Hadoop configuration, placing two map/reduce workers on every node. Verma et al. [2011b] developed an automatic resource inference and allocation framework for MapReduce to meet job deadlines. For the MapReduce placement, they adopted a simple way to assign a fixed number of workers to each node.

Polo et al. [2011] presented a resource-aware scheduler for MapReduce multi-job workloads in which the slot on each node could be dynamically adjusted by leveraging the resource consumptions of different jobs to maximize the resource utilization of the cluster. Later, Polo

et al. [2013] studied deadline-based management for MapReduce workloads based on the same assignment technology of task slots, but here the aim was to ensure the deadline meeting of jobs. Wang et al. [2011] proposed an automatic control mechanism for the dynamic assignment of task slots on each computing node. Using their mechanism, cluster-wide resource utilization was improved.

All this research addresses the MRP problems in non-cloud environments. The objective of these problems is usually to improve the cluster utilization, to reduce the execution time, or to meet the deadline. In addition, in these problems, the total number of machines and the types of machines that can be used for MapReduce computations are given beforehand.

The MRP problem in our study is totally different from those problems in the non-cloud environments, as. the objective of our MRP problem is to minimize the monetary cost of using VMs. Unlike those problems in non-cloud environments, in the MRP problem, the total number and types of the VMsto be used for MapReduce computations are unknown in advance. Instead, we need to select the VM types and the number of the VMs of each selected type, and to determine the placement of workers on the selected VMs. Therefore, the MapReduce placement approaches reviewed for non-cloud-based MapReduce computations cannot be used to address the problem in our study.

2.3.2 Cloud-based MapReduce Placement

Research approaches for the MRP problems in the cloud computing environment that were considered in this review seemed to fall into two main focus areas: direct MRP problems and other placement problems similar to the MRP problem.

In the first grouping, Tian and Chen [2011] worked on minimizing the financial charge for a single MapReduce job while meeting a time deadline, by placing the same number of workers on the same type of slave nodes. AbdelBaky et al. [2012] proposed an objective-driven scheduler which minimized the required number of VMs to meet the deadline constraint for MapReduce-CometCloud. In their scheduler, each VM could load only one mapper or reducer, although the VMs were heterogeneous.

Hwang and Kim [2012] studied a resource provisioning problem for MapReduce in the

cloud that aimed at minimizing the monetary cost of VMs while meeting the deadline constraints. They paid more attention to the placement of the VMs on physical machines; for the problem of map/reduce worker placement on VMs, they adopted a simple way in which a fixed number of map/reduce workers were assigned to each VM.

Lama and Zhou [2012] proposed an automated job provisioning system for Hadoop MapReduce that could automatically configure the number of VMs to achieve the QoS goals while minimizing the incurred cost. But they did not study how to optimize the map/reduce workers on the VMs: they assigned map/reduce workers to the VMs following a basic rule such as having one mapper and one reducer to a small VM and two map workers and two reduce workers to a medium VM.

Chen et al. [2014b] built up a cost function modeling the relationship among execution time, input size, and available cloud resource, and solved a problem aiming at meet deadline requirements with minimum monetary cost. Just like previous research efforts, they studied the optimum number of VMs rather than the placement optimization of map/reduce workers on VMs. With regard to the placement, they placed the same number of map/reduce workers on only one type of VMs.

Unlike the work using the simple rules of assigning workers to VMs, Herodotou et al. [2011] used a more exact method to address the MapReduce placement issue. They developed a system named Elastisizer, included in Starfish, to answer the cluster sizing problems for the MapReduce operated on cloud platforms. This system could tell MapReduce users the best VM type from multiple types provided by public clouds as well as the optimum number of the VMs of that VM type. However, these cluster sizing problems were different from the MRP problem, since the constraints of the cluster sizing problems were to meet the desired requirements on execution time or cost, whereas the constraint of the MRP problem was satisfying the resource requirements of all the workers to be placed. Thus, their approach could not be used to address the MRP problem.

Cardosa et al. [2012] studied how to place the VMs for MapReduce computations on physical machines with minimum energy costs. Their problem was similar to the MRP problem, but the MRP problem was more complicated. For their research, the physical machines or bins were identical, whereas multiple types of VMs or bins were considered in the MRP problem. Also, the number of the bins in their problem was definite, while that number in the MRP problem

was infinite. Thus, their algorithm could not immediately be used to address the MRP problem.

The MapReduce placement approaches involved in the research reviewed can almost be categorized into homogeneous MapReduce placement optimization, as those researchers usually assigned workers on homogeneous VMs or followed homogeneous configurations of worker numbers on each VM. These approaches are totally different from the heterogeneous MapReduce placement optimization approach proposed in our study, which allows using heterogeneous VMs and having heterogeneous placement on each VM used.

In our preliminary work [Xu and Tang, 2014a,b] on the MRP problem, the existing resources (VMs) were not considered, and the algorithm could not make good use of the existing resources to further reduce the cost of cloud-based MapReduce computations. Our current study addresses the issue about how to utilize the existing resources.

In the research reviewed on other placement problems similar to the MRP problem in cloud computing, such as the VM placement problems, these problems are seen as Bin Packing Problems (BPPs), and are solved by modified bin-packing algorithms, such as first-fit-decreasing ([Lee et al., 2011, Verma et al., 2008]), best-fit-decreasing ([Beloglazov and Buyya, 2012]), set covering ([Haouari and Serairi, 2009, Monaci and Toth, 2006]), or other algorithms ([Palanisamy et al., 2011, Srikantaiah et al., 2008]). However, the MRP problem is more complicated than their problems, as it considers both the multiple types of bins (VMs) and the multiple resource constraints, whereas their problems just considered either. Their algorithms therefore cannot be used immediately for the MRP problem.

A number of heuristics have been proposed to solve the BPP and its variants. For instance, several variants of the first-fit-decreasing (FFD) algorithm were proposed to address multi-constraint BBP [Caprara and Toth, 2001, Panigrahy et al., 2011]. In these works, several ways to calculate the surrogate weights were investigated. Kang and Park [2003] presented an iterative FFD (IFFD) especially for the variable sized BPP. In addition, Haouari and Serairi [2009], Monaci and Toth [2006] adopted SCH to solve the multi-constraint and variable sized BPP by transforming the BPP to a set-covering problem. In order to solve the BPP with the variable cost and size, Crainic et al. [2011] proposed an adapted best-fit-decreasing (A-BDF) algorithm which integrated the information of the lower bound computations. Bang-Jensen and Larsen [2012] presented a local search heuristic for the variable sized BPP with a variable bin size in real life, and an important characteristic of this heuristic was it could find solutions within

(milli) seconds. A heuristic inspired from both Minimal Bin Slack and IFFD were proposed to solve the offline variable sized BPP [Maiza et al., 2013].

Besides the heuristic algorithms, meta-heuristic algorithms are also adopted to solve the BPP and its variants. A typical example is the ordering genetic algorithm (OGA) developed by Haouari and Serairi [2009], which was used to address the variable sized BPP. Unlike Haouari's work, Falkenauer [1996] first proposed a grouping genetic algorithm (GGA) to solve the BPP. Iima and Yakawa [2003], Wilcox et al. [2011] respectively modified the original GGA and adopted it to solve the BPP with the same bin size. Brugger et al. [2004] proposed Ant Packing, an ant colony optimization approach, to solve the classical one-dimensional BPP. Liu et al. [2008] designed an evolutionary particle swarm optimization algorithm was adopted to solve the multi-objective BPP. Segura et al. [2011] developed a parallel island-based multi-objective memetic algorithm to address the two-dimensional BBP. Although a number of heuristic and meta-heuristic algorithms have been proposed to solve the BPP and its variants, none of these algorithms is designed for the MRP problem.

2.4 MapReduce Consolidation

The research related to the third problem, the MapReduce consolidation (MRC) problem, is reviewed and compared with our research. Gaps noted in the current literature point to the innovations of our study.

The abundance of research on the consolidation problems in clouds can be categorized into two groups. One has investigated the VM consolidation problems: the consolidation of VMs (equivalent to the workers in the MRC problem) on physical machines (equivalent to the VMs in the MRC problem); the other has studied the MapReduce consolidation problems: the consolidation of MapReduce computations on VMs or physical machines. These two groups of research are each now reviewed and compared with our research.

2.4.1 VM Consolidation

Seven current research projects on VM consolidation problems are reviewed first. Bobroff et al. [2007] investigated the dynamic consolidation of VMs on physical machines to maximize the resource utilization while achieving a certain level of performance. Ferreto et al. [2011]

used several heuristics to solve the server consolidation problem of minimizing the number of physical machines.Lin et al. [2011] investigated the VM consolidation on physical machines to minimize the number of physical machines used while meeting the resource requirements of VMs. Wu et al. [2012] developed a genetic algorithm to consolidate VMs on physical machines with the minimum of the physical machines. Beloglazov and Buyya [2013] also investigated how to manage overloaded machines for dynamic VM consolidation under QoS constraints. Although all of these VM consolidation problems were solved by various methods, they can be categorized into the same group.

With the research into these problems, the physical machines were assumed to be identical while the VMs could be various and had no certain types. The MRC problem is different from these VM consolidation problems, as it assumes multiple types of workers and multiple VMs. It is much more complicated than these VM consolidation problems, so none of their various methods can be used for the MRC problem.

A more complicated VM consolidation problem investigated by Li et al. [2013] was working on how to consolidate heterogeneous workloads on heterogeneous physical machines. They proposed two consolidation algorithms to execute all required VMs with the minimum number of physical machines. Their objective was to minimize the energy consumption. However, their algorithms cannot be used for the MRC problem, with its goal of minimizing both the cost of using VMs and the migration cost.

2.4.2 MapReduce Consolidation

The literature review covering the consolidation problems related to MapReduce computations focuses on three main research studies, and gives summary notes regarding several other projects.

Huang et al. [2012] studied the consolidation of the VMs for MapReduce computations and the VMs for non-MapReduce computations on physical machines, and proposed a heuristic to minimize the usage of physical machines while meeting the resource requirements of MapReduce computations. The problem for these researchers was that the resource capacities of the physical machines (equivalent to the VMs in the MRC problem) were heterogeneous, since the physical machines also provided resources for those non-MapReduce computations. They also assumed that the VMs (equivalent to the workers in the MRC problem) to be consolidated were

identical. The complexity of our MRC problem, with its multiple types of VMs and workers, and its varied numbers of each type of worker, means that it cannot be solved by their method.

Cardosa et al. [2012] studied how to consolidate the VMs for MapReduce computations on physical machines with minimum energy costs, and proposed several placement algorithms to solve their problem. They also assumed that the physical machines (equivalent to the VMs in the MRC problem) were identical, which simplified their research problem. As our MRC problem tackles heterogeneous VMs, it unable to use their proposed algorithms.

Palanisamy et al. [2014], who investigated how to consolidate MapReduce computations on VMs with the minimum cost of using VMs, developed an online scheduling algorithm to solve that problem. Unlike the other research discussed here, their problem allowed heterogeneous VMs. But they made assumptions, when simplifying their problem, that each MapReduce computation was processed by a dedicated cluster of VMs of the same type and that one VM could not process multiple MapReduce computations at the same time. In the MRC problem, multiple MapReduce computations can share one VM and one MapReduce computation can be processed by multiple types of VMs at the same time. Hence, the MRC problem is totally different from theirs, so their algorithm cannot be used to solve it.

Other research studied several problems related to MapReduce consolidation in non-cloud environments. The objectives of these various problems included maximizing cluster utilization [Polo et al., 2011, Wang et al., 2011], minimizing execution time [Herodotou and Babu, 2011, Lin et al., 2010b, Wolf et al., 2010, Xie et al., 2010, Zaharia et al., 2008], and minimizing energy consumption [Lang and Patel, 2010, Maheshwari et al., 2012]. Unlike those problems, the objective of the MRC problem is to minimize the operational cost of cloud-based MapReduce. Thus, their approaches also cannot be used directly for our MRC problem.

To the best of our knowledge, our research is the first attempt to solve our MRC problem. This problem, which is different from or even more complicated than the consolidation problems studied in other current research, seems not to be able to be solved by the current methods outlined from the review. Therefore, our research will propose a new consolidation algorithm to solve the MRC problem.

2.5 Resource Provisioning Frameworks for Cloud-based MapReduce

In general, the resource provisioning frameworks for cloud-based MapReduce can be divided into three types: *static/conservative resource provisioning frameworks*, *static/accurate resource provisioning frameworks* and *dynamic resource provisioning frameworks*. This section reviews how these three frameworks have been developed and adopted in the current research, affecting the resource provisioning problems for cloud-based MapReduce and motivating the research direction of our study towards a new type of framework.

2.5.1 Static/Conservative Resource Provisioning Frameworks

Some current research has adopted static/conservative resource provisioning frameworks for the resource provisioning problems in cloud-based MapReduce. Two research examples are reviewed to show how these frameworks have been adopted.

Verma et al. [2011b] proposed a resource provisioning framework for MapReduce computations with performance goals. In this framework, the upper bound of the MapReduce completion time was estimated and the minimum resource amount to meet performance goals was deducted. The resource was also pre-determined and could not be adjusted at runtime.

Palanisamy et al. [2014] proposed a resource provisioning framework, called Cura, that aimed to minimize the resource usage costs of cloud-based MapReduce computations while meeting deadlines. Their resource allocation for a MapReduce computation was pre-determined and could not be changed at runtime. The allocation amount was based on the worst estimation for the MapReduce performance so as to meet deadlines.

With this sort of framework, the resource provisioning is pre-determined and cannot be changed at runtime. To guarantee the QoS of MapReduce, the performance of MapReduce computations is estimated under a worst-case scenario, and extra resources are provisioned in case of unexpected performance degradation. This type of framework is easy to implement and has the ability of QoS-guarantee even under a dynamical environment, but it sacrifices the operational cost of MapReduce computations for the QoS-guarantee. When variation range in the dynamical environment is larger, the cost of QoS-guarantee is even higher. Thus, this sort of frameworks can hardly achieve the objective of cost minimization of the QoS-guaranteed resource provisioning problem motivating this study, and so cannot address this problem well.

Our framework is different from the static/conservative resource provisioning frameworks. It supports resource scaling-down at runtime. Once it finds the computation running ahead of the schedule due to the environment changes, extra resources can be removed such that the cost of extra resources can be saved. Therefore, compared with the static/conservative frameworks, our framework is more cost-effective, and more suitable for the QoS-guaranteed resource provisioning problem.

2.5.2 Static/Accurate Resource Provisioning Frameworks

Other researchers have developed static/accurate resource provisioning frameworks to use for the resource provisioning problems in cloud-based MapReduce.

Tian and Chen [2011] and Chen et al. [2014b] proposed resource provisioning frameworks for guaranteeing the deadlines of cloud-based MapReduce computations. Their frameworks were designed based on the exact cost function models which quantified the relationship among execution time, input size and available cloud resource, and could accurately estimate the required amount of resource provisioning for meeting deadlines. Their frameworks are effective on meeting deadlines in stable computational environments. However, when unexpected performance degradation happens on VMs or network, their frameworks cannot add resource provisioning in time, leading to deadline misses.

Using machine learning technologies, Herodotou et al. [2011] developed a resource provisioning framework named Elastisizer. This framework determined the exact cluster size for cloud-based MapReduce computations to meet a certain level of performance requirements such as execution time. Similarly, through machine learning performance modeling, Lama and Zhou [2012] proposed an automated resource provisioning framework for cloud-based MapReduce computations to meet deadlines with minimum resource costs. However, both these frameworks do not support resource scaling and cannot scale up resource provisioning when current resource provisioning is insufficient to meet deadlines.

Like the static/conservative ones, this type of framework also pre-determines the resource provisioning and cannot change the resource provisioning at runtime, but tries to estimate exactly the amount of the resource required to guarantee the QoS such it can save more computation cost. Compared with the static/conservative frameworks, this sort of framework reduces

more cost of MapReduce computations, but probably violates the QoS constraint in the QoS-guaranteed resource provisioning problem when the performance degradation occurs. Thus, this type of framework cannot effectively address our study's problem.

Unlike these static/accurate resource provisioning frameworks, our framework supports resource scaling-up during runtime. Once the computation running behind the schedule is detected, the framework scales up resources to speed up the computation, to guarantee the QoS. Therefore, compared with the static/accurate resource provisioning frameworks, our framework has the capacity of QoS-guarantee, and is more suitable for our research problem.

2.5.3 Dynamical Resource Provisioning Frameworks

The drawbacks of these static resource provisioning frameworks have led to the proposal of dynamic resource provisioning frameworks that allow the adjustment of resource provisioning at runtime. To minimize cost while meeting application deadlines in cloud workflows, Mao and Humphrey [2011] proposed a resource scaling mechanism that is implemented repeatedly using a monitor control loop. In each loop, scheduling/scaling decisions were made according to updated progress information. Similarly, Petrucci et al. [2011] adopted an optimization control loop of a certain time to minimize the power consumption, while meeting performance requirements of heterogeneous and changing workloads in a virtualized data center. But both of these two frameworks were not designed specially for MapReduce computations, so their frameworks could not immediately solve our study problem.

The following research projects developed the dynamical resource provisioning frameworks specially for MapReduce. Mattess et al. [2013] designed a resource scaling framework to periodically adjust the resource size. However, their framework met only the soft deadlines of MapReduce computations. AbdelBaky et al. [2012] searched for the required number of VMs to meet the deadline of MapReduce-CometCloud, and adopted a periodic scheduling framework to estimate that required number. But their framework was designed for meeting soft deadlines, not for meeting hard deadlines.

Cardosa et al. [2011] designed a resource provisioning framework supporting the addition of VMs during the map phases of MapReduce computations to meet the deadlines. But their framework could hardly guarantee the hard deadlines of MapReduce computations, especially when performance degradation occurred in the reduce phases.

Rao and Reddy [2012] proposed a framework which determined the required number of Map/Reduce slots for every computation to meet the deadline while maximizing the data locality and resource utilization. However, their experimental results showed that a MapReduce computation named Permutation Generator actually missed the deadline using their framework. Thus, their framework did not always meet the deadline constraints of MapReduce computations. In comparison, the theoretical results presented in this study prevent deadline misses in theory, as will be demonstrated in later experiments.

These resource provisioning frameworks are all periodic ones which periodically scale up/down resource provisioning. However, the determination of an optimum period is very difficult. Different periods incur different costs of using VMs and have different performances on QoS-guarantee. In addition, the periodic frameworks in the current literature cannot always meet QoS, the hard deadline of cloud-based MapReduce. Therefore, this sort of frameworks is ineffective for the QoS-guaranteed resource provisioning problem.

To solve this problem, our research develops a new type of dynamic resource provisioning frameworks: event-driven resource provisioning. It also supports resource scaling at runtime to guarantee QoS while minimizing costs. But, unlike the periodic frameworks periodically activating resource provisioning, the new framework uses an event-driven mechanism, making the amount and timing of resource provisioning more accurate. Moreover, the event-driven resource provisioning in the new framework is based on the solid theoretical analysis of QoS-guarantee of cloud-based MapReduce. So it is more effective than the periodic frameworks for the QoS-guaranteed resource provisioning problem.

2.6 Summary of Chapter

This chapter has reviewed the literature related to the QoS-guaranteed resource provisioning of cloud-based MapReduce, and then has identified the gaps in current literatures and finally has presented the motivation and uniqueness of the research in this project.

First of all, the review of the state of the art of the QoS-guaranteed resource provisioning for cloud-based MapReduce showed that the QoS-guaranteed resource provisioning problems addressed by other research are different from the problem presented in this study. Our QoS-guaranteed resource provisioning for cloud-based MapReduce is a totally new issue and needs

to be solved. After that, our four research problems have been reviewed separately. The research work on our four research problems is different from that presented by other work. Thus, this research gives the details on the four research problems and finally solves the QoS-guaranteed resource provisioning problem for cloud-based MapReduce.

Chapter 3

Theoretical Results of QoS-guaranteed Resource Provisioning

To solve the QoS-guaranteed resource provisioning for cloud-based MapReduce, a theoretical study needs to be conducted. The results from the theoretical study will be applied in the development of the new QoS-guaranteed resource provisioning framework. The theoretical study is presented in the following sections. Section 3.1 gives the introduction of theoretical study. Section 3.2 models the impact of resource scaling on MapReduce computation time. Section 3.3 presents the problem description. Section 3.4 deducts the theoretical results for QoS-guaranteed resource provisioning for cloud-based MapReduce. Section 3.5 demonstrates the theoretical results by case studies. Finally, Section 3.6 presents the summary of this chapter.

3.1 Introduction

This chapter presents a detailed theoretical study of the QoS-guaranteed resource provisioning for cloud-based MapReduce. The theoretical results, summarized in two corollaries and three theorems, answer the research questions directly or partially. The two corollaries are used to judge if current resource provisioning is sufficient to meet the deadlines of cloud-based MapReduce computations or not, which answers the research question regarding how to determine the timing of resource provisioning for cloud-based MapReduce computations. The three theorems quantify the sufficient conditions on the amount and timing of resource provisioning for meeting the deadlines of cloud-based MapReduce computations, which answer the research questions of how to determine the amount and timing of resource provisioning for cloud-based MapReduce

computations.

The main contributions of this chapter are summarized as follows in answering these research questions:

1. A nonlinear transformation is employed to define the problem in a *reverse resource space*, simplifying theoretical analysis significantly;
2. Sufficient conditions for QoS-guaranteed scaling-up/down for MapReduce computation are established; and
3. The latest intervention time after which QoS cannot be guaranteed by resource scaling-up is derived.

3.2 Impact of Resource Scaling on MapReduce Computation Time

As shown in Figure 3.1, the MapReduce computation time consists of two parts, the computation time of map phase and the computation time of reduce phase. During the map/reduce phase, the map/reduce tasks are executed in parallel. When map/reduce tasks are more than map/reduce workers, the map/reduce tasks are completed in several waves. When the number of map/reduce tasks are equal to the number of map/reduce workers, and the map/reduce tasks are completed in one wave. In addition, there is an overlap between the map and reduce computations because the first wave of reduce tasks is usually executed in parallel with the map phase. Thus, the computation time of the overlapping parts of the reduce tasks should be excluded from the total MapReduce computation time. For simplicity, the non-overlapping part of a reduce task in the first wave is taken as a complete reduce task. In other words, the computation time of a reduce task in the first wave amounts to the computation time of the non-overlapping part of that reduce task.

Next, the impact of resource scaling on MapReduce computation time will be quantified. Let us start with the following lemma, which is the Makespan theorem [Verma et al., 2011a] for estimating the upper bound of the completion time of n tasks ($n \geq 1$) executed on N computing nodes in parallel.

Lemma 1 *The completion time of all the n tasks running in parallel achieves maximal when the first $n-1$ tasks are completed at the same time and the execution time of the last task*

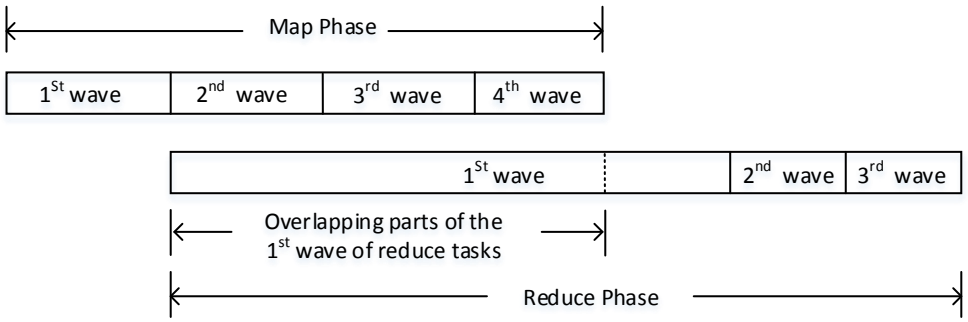


Figure 3.1: Process of a typical MapReduce computation

achieves maximal. Let T^{max} denote the maximal execution time of the tasks. Then the upper bound of the completion time of all the n tasks is expressed as $(n-1)T^{avg}/N + T^{max}$.

3.2.1 Resource Scaling-up

How to calculate the upper bound of the computation time of a MapReduce computation after resource scaling-up is investigated here. Let us say a MapReduce computation consists of m map tasks and r reduce tasks. The computation is pre-provisioned with M_0 map workers and R_0 reduce workers. At time $t \geq 0$, M_+ map workers and R_+ reduce workers are added to execute the remaining tasks of the MapReduce computation. At this moment, the numbers of the remaining map and reduce tasks are denoted as m_t and r_t, respectively, which can be calculated as follows:

$$m_t = \sum_{i=1}^{m} p_{it}^m \tag{3.1}$$

$$r_t = \begin{cases} r, & m_t > 0 \\ \sum_{j=1}^{r} p_{jt}^r, & m_t = 0 \end{cases} \tag{3.2}$$

where p_{it}^m (p_{jt}^r) denotes the ratio of the size of the map (reduce) input without being processed to the total input size for the i^{th} map task (the j^{th} reduce task).

A simple example is given to show how to use Equations (3.1) and (3.2) to calculate m_t and r_t. Consider three map tasks: a completed one ($p_{1t}^m = 1$), a running one which has processed its half input ($p_{2t}^m = 0.5$), and a pending one ($p_{3t}^m = 0$). According to Equation (3.1), $m_t = (1-1) + (1-0.5) + (1-0) = 1.5$. r_t is calculated in the same way; but when $m_t > 0$, the map phase is not completed and no reduce tasks start running, suggesting $r_t = r$.

Let T_o be the time delay for launching the new map/reduce workers. During the time period,

T_o, m_o map tasks and r_o reduce tasks are completed. The lower bounds of m_o and r_o, m_o^{min} and r_o^{min}, are calculated as follows:

$$m_o^{min} = \frac{T_o}{T_m^{max}} M_0 \tag{3.3}$$

$$r_o^{min} = \begin{cases} 0, & m_t > 0 \\ \frac{T_o}{T_r^{max}} R_0, & m_t = 0 \end{cases} \tag{3.4}$$

In Equation (3.3), T_o/T_m^{max} calculates the minimum number of the map tasks completed by one mapper during the time period T_o. As there are M_0 map workers to execute the tasks simultaneously, the lower bound of m_o, m_o^{min}, is equal to $M_o \cdot T_o/T_m^{max}$. The lower bound of r_o, r_o^{min}, is calculated in a similar way. When $m_t > 0$, as no reduce tasks start running, $r_o^{min} = 0$.

Consequently, the upper bound of the computation time of completing the remaining map tasks, $T_{m+}^{up}(M_+)$, is derived from Lemma 1, which is expressed by

$$T_{m+}^{up}(M_+) = \begin{cases} \frac{(m_t - m_o^{min} - 1)T_m^{avg}}{M_0 + M_+} + T_m^{max}, & m_t \geq m_o^{min} + 1 \\ \alpha T_m^{max}, & m_t < m_o^{min} + 1 \end{cases} \tag{3.5}$$

where α is a binary number, and $\alpha = 0$ if $m_t \leq m_o$, T_m^{avg} and T_m^{max} represent the average and maximum computation times of a map task, respectively. When $m_t < m_o^{min} + 1$, fewer than one map task are left, so $T_{m+}^{up}(M_+) = T_m^{max}$. When $m_t \leq m_o^{min}$, no map tasks are left, so $T_{m+}^{up}(M_+) = 0$.

Similarly, the upper bound of the computation time of completing the remaining $r_t - r_o$ reduce tasks, $T_{r+}^{up}(R_+)$, is derived from Lemma 1, which is given by

$$T_{r+}^{up}(R_+) = \begin{cases} \frac{(r_t - r_o^{min} - 1)T_r^{avg}}{R_0 + R_+} + T_r^{max}, & r_t \geq r_o^{min} + 1 \\ \beta T_r^{max}, & r_t < r_o^{min} + 1 \end{cases} \tag{3.6}$$

where β is a binary number, and $\beta = 0$ if $r_t \leq r_o^{min}$, T_r^{avg} and T_r^{max} respectively represent the average and maximum computation times of a reduce task.

With Equation (3.6), the upper bound of the MapReduce computation time after M_+ map workers and R_+ reduce workers are added at the time t is given by

$$T_+^{up}(M_+, R_+) = t + T_o + T_{m+}^{up}(M_+) + T_{r+}^{up}(R_+) \tag{3.7}$$

3.2.2 Resource Scaling-down

The upper bound of the MapReduce computation time after resource scaling-down is calculated as follows. The MapReduce computation with m map tasks and r reduce tasks is pre-provisioned with M_0 map workers and R_0 reduce workers. At the time $t \geq 0$, M_- map workers and R_- reduce workers are removed from the computation. Meanwhile, there are m_t and r_t reduce tasks left, which are calculated from Equation (3.1).

The upper bound of the computation time of the remaining m_t map tasks, $T_{m-}^{up}(M_-)$, is derived from Lemma 1. It is expressed as

$$T_{m-}^{up}(M_-) = \begin{cases} \frac{(m_t-1)T_m^{avg}}{M_0-M_-} + T_m^{max} - T_m^{avg}, & m_t > 0 \\ 0, & m_t = 0 \end{cases} \tag{3.8}$$

When $m_t = 0$, $T_{m-}^{up}(M_-) = 0$ as no map tasks are left.

Similarly, the upper bound of the computation time of the remaining r_t reduce tasks, $T_{r-}^{up}(R_-)$ is derived from Lemma 1. It is given by

$$T_{r-}^{up}(R_-) = \begin{cases} \frac{(r_t-1)T_r^{avg}}{R_0-R_-} + T_r^{max} - \gamma T_r^{avg}, & r_t > 0 \\ 0, & r_t = 0 \end{cases} \tag{3.9}$$

where γ is a binary number, and $\gamma = 0$ if $m_t > 0$. When $r_t = 0$, no reduce tasks are left and thus $T_{r-}^{up}(R_-) = 0$.

With the results in Equation (3.9), the upper bound of the MapReduce computation time after M_- map workers and R_- reduce workers are removed at the time t is given by

$$T_-^{up}(M_-, R_-) = t + T_{m-}^{up}(M_-) + T_{r-}^{up}(R_-) \tag{3.10}$$

3.3 Problem Description

The theoretical study of QoS-guaranteed resource provisioning for cloud-based MapReduce consists of three issues: the *scaling up issue*, the *latest intervention time issue* and the *scaling down issue*. An example of hard real-time applications in cloud-based MapReduce is used to

illustrate the existence of these three issues.

The scaling-up issue is raised when the pre-provisioned resource is insufficient to meet the deadlines of hard real-time applications. If there is a performance degradation on the VMs or network, the computation progress of the applications will fall behind the schedule. Without resource scaling-up, the deadlines will be missed, leading to a system failure or even a disaster. To avoid deadline misses, it is necessary to derive a scaling-up solution (M_+, R_+) at time $t \geq 0$ so that the total computation time $T_+(M_+, R_+) \leq D$, where D is the deadline of the MapReduce computation.

The latest intervention time issue characterizes the situation in which the resource scaling-up is activated too late to meet the MapReduce deadlines. In this case, no pending tasks are left and consequently any additional map/reduce workers do not help speed up the computation. To avoid this situation, the latest intervention time needs to be determined in advance, giving the time before which there exists at least one scaling-up solution (M_+, R_+) to ensure the total computation time $T_+(M_+, R_+) \leq D$.

The scaling-down issue describes the scenario when the pre-provisioned resource is over-provisioned for guaranteeing MapReduce deadlines. If there is a performance promotion on the VMs or network, the computation progresses ahead of the schedule. To minimize resource consumption, the resource provisioning needs to be scaled down. This requires determining a scaling-down solution (M_-, R_-) at the runtime $t \geq 0$ so that the total computation time $T_-(M_-, R_-) \leq D$.

3.3.1 Reverse Resource Space

To simplify the mathematical operations in our theoretical analysis, the QoS-guaranteed resource scaling problem is re-defined in a new space, the *Reverse Resource Space*, through the following nonlinear maps:

$$\mathrm{F}_+ : \mathbb{M}_+ \to \frac{1}{M_0 + M_+},\ \mathbb{R}_+ \to \frac{1}{R_0 + R_+} \tag{3.11}$$

$$\mathrm{F}_- : \mathbb{M}_- \to \frac{1}{M_0 - M_-},\ \mathbb{R}_- \to \frac{1}{R_0 - R_-} \tag{3.12}$$

Definition 1 (Reverse Resource Space) *The reverse resource space is a Euclidean plane transformed from the two-dimensional Euclidean Resource Space using* F_+ *and* F_-.

Through the nonlinear map F_+, the scaling-up solution (M_+, R_+) in the original resource space is mapped to the scaling-up solution $(\mathbb{M}_+, \mathbb{R}_+)$ in the reverse resource space, as shown in Figure 3.2. The notation $\mathbb{M}_+$ is the reciprocal of the amount of the mapper provisioning after resource scaling-up, $1/(M_0 + M_+)$, while the notation $\mathbb{R}_+$ is the reciprocal of the amount of the reduce worker provisioning after resource scaling-up, $1/(R_0 + R_+)$.

Meanwhile, through the nonlinear map F_-, the scaling-down solution (M_-, R_-) in the original resource space is mapped to the scaling-down solution $(\mathbb{M}_-, \mathbb{R}_-)$ in the reverse resource space, as shown in Figure 3.3. The notation $\mathbb{M}_-$ is the reciprocal of the amount of the mapper provisioning after resource scaling-down, $1/(M_0 - M_-)$, while the notation $\mathbb{R}_-$ is the reciprocal of the amount of the reduce worker provisioning after resource scaling-down, $1/(R_0 - R_-)$.

The QoS-guaranteed resource scaling problem is re-defined in the reverse resource space as follows. Let $T^*_+(\mathbb{M}_+, \mathbb{R}_+)$ and $T^*_-(\mathbb{M}_-, \mathbb{R}_-)$ respectively denote the expressions of $T_+(M_+, R_+)$ and $T_-(M_-, R_-)$ in the reverse resource space. Then the scaling-up issue is to determine a scaling-up solution $(\mathbb{M}_+, \mathbb{R}_+)$ such that $T^*_+(\mathbb{M}_+, \mathbb{R}_+) \leq D$ holds. The latest intervention time issue is to determine the time before which there exists one scaling-up solution $(\mathbb{M}_+, \mathbb{R}_+)$ to ensure $T^*_+(\mathbb{M}_+, \mathbb{R}_+) \leq D$. The scaling-down issue is to derive a scaling-down solution $(\mathbb{M}_-, \mathbb{R}_-)$ to guarantee $T^*_-(\mathbb{M}_-, \mathbb{R}_-) \leq D$.

In the original resource space, the upper bound of the MapReduce computation time after adding M_+ map workers and R_+ reduce workers, $T^{up}_+(M_+, R_+)$, is presented in Equation (3.7); in the reverse resource space, $T^{up}_+(M_+, R_+)$ is indicated by $T^{up*}_+(\mathbb{M}_+, \mathbb{R}_+)$, which is presented in Equation (3.13).

$$T^{up*}_+(\mathbb{M}_+, \mathbb{R}_+) = t + T_o + T^{up*}_{m+}(\mathbb{M}_+) + T^{up*}_{r+}(\mathbb{R}_+) \tag{3.13}$$

In Equation (3.13), $T^{up*}_{m+}(\mathbb{M}_+)$ and $T^{up*}_{r+}(\mathbb{R}_+)$ are respectively expressed by

$$T^{up*}_{m+}(\mathbb{M}_+) = \begin{cases} (m_t - m_o^{min} - 1)T_m^{avg}\mathbb{M}_+ + T_m^{max}, & m_t \geq m_o^{min} + 1 \\ \alpha T_m^{max}, & m_t < m_o^{min} + 1 \end{cases} \tag{3.14}$$

$$T^{up*}_{r+}(\mathbb{R}_+) = \begin{cases} (r_t - r_o^{min} - 1)T_r^{avg}\mathbb{R}_+ + T_r^{max}, & r_t \geq r_o^{min} + 1 \\ \beta T_r^{max}, & r_t < r_o^{min} + 1 \end{cases} \tag{3.15}$$

Similarly, in the original resource space, the upper bound of the MapReduce computation time after removing M_- map workers and R_- reduce workers, $T_-^{up}(M_-, R_-)$, is presented in Equation (3.7); in the reverse resource space, it is indicated by $T_-^{up*}(\mathbb{M}_-, \mathbb{R}_-)$, which is presented in Equation (3.16).

$$T_-^{up*}(\mathbb{M}_-, \mathbb{R}_-) = t + T_{m-}^{up*}(\mathbb{M}_-) + T_{r-}^{up*}(\mathbb{R}_-) \tag{3.16}$$

In Equation (3.16), $T_{m-}^{up*}(\mathbb{M}_-)$ and $T_{r-}^{up*}(\mathbb{R}_-)$ are respectively expressed by

$$T_{m-}^{up*}(\mathbb{M}_-) = \begin{cases} (m_t + M_0 - 1)T_m^{avg}\mathbb{M}_- + T_m^{max} - T_m^{avg}, & m_t > 0 \\ 0, & m_t = 0 \end{cases} \tag{3.17}$$

$$T_{r-}^{up*}(\mathbb{R}_-) = \begin{cases} (r_t + \gamma R_0 - 1)T_r^{avg}\mathbb{R}_- + T_r^{max} - \gamma T_r^{avg}, & r_t > 0 \\ 0, & r_t = 0 \end{cases} \tag{3.18}$$

Remark 1 *$T_+^{up*}(\mathbb{M}_+, \mathbb{R}_+)$ and $T_-^{up*}(\mathbb{M}_-, \mathbb{R}_-)$ are two linear functions.*

Through this nonlinear transformation, many nonlinear relationships in the original problem are reduced to linear relationships, which greatly simplifies our theoretical analysis. In the rest of this chapter, all discussions about the resource scaling problem are in the reverse resource space unless otherwise specified explicitly.

3.4 Theoretical Analysis of QoS-guaranteed Resource Scaling

The theoretical results of QoS-guaranteed resource scaling are summarized in three theorems and two corollaries. They are presented below in detail.

3.4.1 Theoretical Analysis of the Scale-up Issue

The *QoS-guaranteed Scaling-up* theorem is derived to characterize the conditions for resource scaling-up. It is used to help the resource provisioning framework to determine how much resource needs to be scaled up. It is formally given as follows.

Theorem 1 (QoS-guaranteed Scaling-up) *The scaling-up solution* $(\mathbb{M}_+, \mathbb{R}_+)$ *is a QoS-guaranteed scaling-up solution if the condition* (3.19) *is satisfied when* $m_t \geq m_o^{min} + 1$ *or the condition* (3.25) *is satisfied when* $m_t \leq m_o^{min} + 1$ *and* $r_t \geq r_o^{min} + 1$.

$$a\mathbb{M}_+ + b\mathbb{R}_+ \leq c,\ 0 < \mathbb{M}_+ \leq \frac{1}{M_0},\ 0 < \mathbb{R}_+ \leq \frac{1}{R_0}, \tag{3.19}$$

$$0 < \mathbb{M}_+ \leq \frac{1}{M_0},\ 0 < \mathbb{R}_+ \leq \min\left\{\frac{d}{b}, \frac{1}{R_0}\right\}, \tag{3.20}$$

where a, b, c, *and* d *are respectively expressed by the following equations:*

$$a = (m_t - m_o^{min} - 1)T_m^{avg}, \tag{3.21}$$

$$b = (r_t - r_o^{min} - 1)T_r^{avg}, \tag{3.22}$$

$$c = D - t - T_o - T_m^{max} - T_r^{max}, \tag{3.23}$$

$$d = D - t - T_o - \alpha T_m^{max} - T_r^{max}, \tag{3.24}$$

where α *is a binary number, and* $\alpha = 0$ *if* $m_t \leq m_o^{min}$.

Proof When $T_+^{up*}(\mathbb{M}_+, \mathbb{R}_+) \leq D$ holds, the inequality $T_+(\mathbb{M}_+, \mathbb{R}_+) \leq D$ is true, suggesting that QoS is guaranteed. Then if $m_t \geq m_o^{min}+1$ and $r_t \geq r_o^{min}+1$, according to Equations (3.13), (3.14) and (3.15),

$$\begin{aligned} & T_+^{up*}(\mathbb{M}_+, \mathbb{R}_+) \leq D \\ \Leftrightarrow\ & t + T_o + a\mathbb{M}_+ + T_m^{max} + b\mathbb{R}_+ + T_r^{max} \leq D \\ \Leftrightarrow\ & a\mathbb{M}_+ + b\mathbb{R}_+ \leq c. \end{aligned}$$

In this case, $r_t = r$ and $r_o^{min} = 0$, and consequently $r_t = r < 1$. Also, as no reduce task starts when $m_t > 0$, we have $r_t = r \geq 1$. Therefore, the situation when $m_t \geq m_o^{min} + 1 > 0$ and $r_t < r_o^{min} + 1$ does not exist.

If $m_t < m_o^{min} + 1$ and $r_t \geq r_o^{min} + 1$, we have

$$\begin{aligned} & T_+^{up*}(\mathbb{M}_+, \mathbb{R}_+) \leq D \\ \Leftrightarrow\ & t + T_o + \alpha T_m^{max} + b\mathbb{R}_+ + T_r^{max} \leq D \\ \Leftrightarrow\ & \mathbb{R}_+ \leq \frac{d}{b}. \end{aligned}$$

If $m_t < m_o^{min}+1$ and $r_t < r_o^{min}+1$, the computation finishes when new map/reduce workers start to work. As we consider only the resource scaling for a running MapReduce computation, the situation when $m_t < m_o^{min} + 1$ and $r_t < r_o^{min} + 1$ is not considered.

Furthermore, we have $M_+, R_+ \geq 0$, $0 < \mathbb{M}_+ \leq 1/M_0$, $0 < \mathbb{R}_+ \leq 1/R_0$. Consequently, the QoS-guaranteed scaling-up conditions (3.19) and (3.25) are derived. ∎

Remark 2 *The QoS-guaranteed scaling-up condition* (3.19) *is a sufficient condition, not a necessary one.*

The above remark indicates that when the condition is met, the QoS is guaranteed. However, there might be some occasions when the condition is not met but the QoS may still be met.

Remark 3 *Both cases of the QoS-guaranteed scaling-up condition* (3.19) *are expressed in linear inequalities in the reverse resource space. This significantly simplifies our mathematical operations in theoretical analysis.*

The following remark explains the area in Figure 3.2.

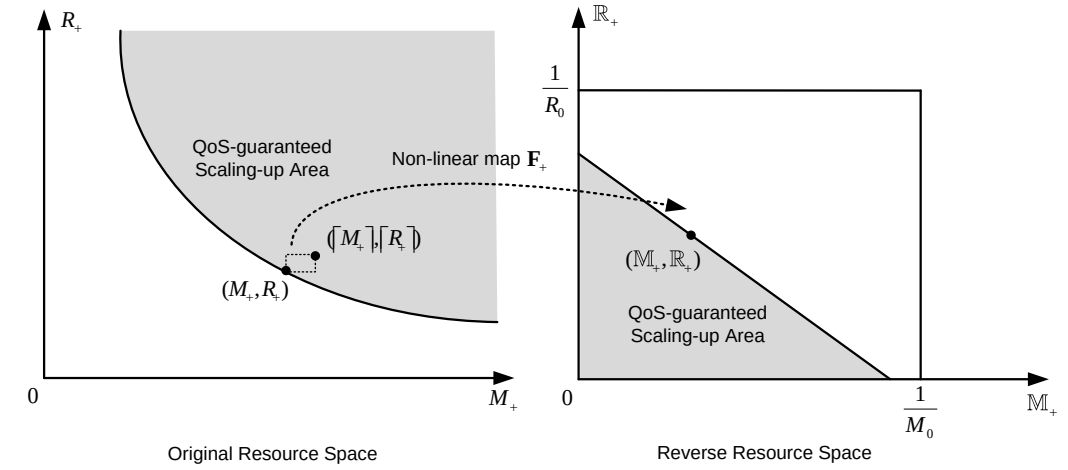
Remark 4 *Each of the two cases of the QoS-guaranteed scaling-up condition* (3.19) *indicates an area that is referred to as the QoS-guaranteed Scaling-up Area.*

The following remark describes how a point in the reverse resource space is re-mapped to a solution in the original resource space, as shown in Figure 3.2.

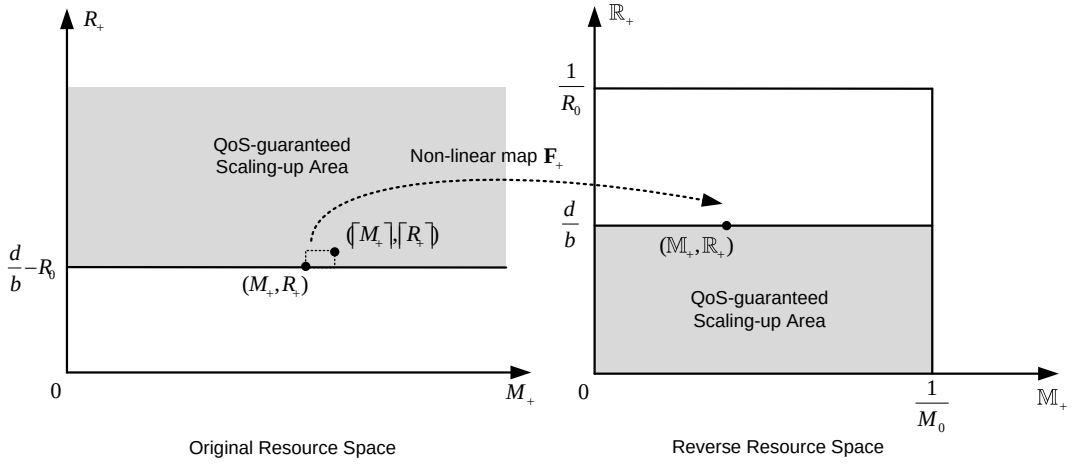
Remark 5 *Any point* $(\mathbb{M}_+, \mathbb{R}_+)$ *in the QoS-guaranteed scaling-up area is re-mapped to a QoS-guaranteed solution* $(\lceil M_+ \rceil, \lceil R_+ \rceil)$ *in the original resource space by the non-linear maps* F_+*, where* $M_+ = 1/\mathbb{M}_+ - M_0$ *and* $R_+ = 1/\mathbb{R}_+ - R_0$*.*

The *Resource Scaling-up Condition* corollary is derived from the above theorem. The following corollary is used to help the resource provisioning framework judge whether the resource scaling-up needs to be done.

Corollary 1 (Resource Scaling-up Condition) *If the scaling-up solution* $(1/M_+, 1/1/R_+)$ *is not in the QoS-guaranteed scaling-up area, resource scaling-up is required.*



(a) $m_t \geq m_o^{min} + 1$



(b) $m_t \leq m_o^{min} + 1$ and $r_t \geq r_o^{min} + 1$

Figure 3.2: QoS-guaranteed scaling-up areas

Proof The scaling-up solution $(1/M_+, 1/1/R_+)$ is equivalent to the solution $(0, 0)$ in the original resource space, implying that no map/reduce worker is scaled up. If $(1/M_+, 1/1/R_+)$ is not in the QoS-guaranteed scaling-up area, the condition (3.19) or (3.25) will not be satisfied and the total MapReduce computation time may exceed the deadline. In this case, resource needs to be scaled up to guarantee the QoS deadline. ∎

3.4.2 Theoretical Analysis of the Latest Intervention Time Issue

The *Latest Intervention Time* theorem is derived to characterize the condition on the time point of resource scaling-up for guaranteeing QoS. It addresses how to determine the time of resource scaling-up when current resource provisioning is insufficient to guarantee QoS. It is formally given as follows.

Theorem 2 (Latest Intervention Time) *At least one QoS-guaranteed scaling-up solution exists if the time point of resource scaling-up satisfies the following condition* (3.25)*.*

$$t < D - T_o - T_m^{max} - T_r^{max} \tag{3.25}$$

where t is the time point of resource scaling-up, D is the deadline, T_o is the time delay of resource scaling-up, T_m^{max} and T_r^{max} respectively denote the maximum computation times of map and reduce tasks.

Proof When $m_t \geq m_o^{min} + 1$,

$$T_+^{up*}(\mathbb{M}_+, \mathbb{R}_+) = t + T_o + a\mathbb{M}_+ + T_m^{max} + b\mathbb{R}_+ + T_r^{max}.$$

Moreover,

$$\begin{aligned} & 0 < \mathbb{M}_+ \leq \frac{1}{M_0}, 0 < \mathbb{R}_+ \leq \frac{1}{R_0} \\ \Rightarrow \quad & T_+^{up*}(\mathbb{M}_+, \mathbb{R}_+) > t + T_o + T_m^{max} + T_r^{max}. \end{aligned}$$

Then if $t + T_o + T_m^{max} + T_r^{max} < D$, there exists at least one scaling-up solution to ensure that the inequality $T_+^{up*}(\mathbb{M}_+, \mathbb{R}_+) < D$ holds. In addition,

$$\begin{aligned} & t < D - T_o - T_m^{max} - T_r^{max} \\ \Leftrightarrow \quad & t + T_o + T_m^{max} + T_r^{max} < D. \end{aligned}$$

When $m_t < m_o^{min} + 1$ and $r_t \geq r_o^{min} + 1$,

$$T_+^{up*}(\mathbb{M}_+, \mathbb{R}_+) = t + T_o + \alpha T_m^{max} + b\mathbb{R}_+ + T_r^{max}.$$

Also,

$$\begin{aligned} & 0 < \mathbb{M}_+ \leq \frac{1}{M_0}, 0 < \mathbb{R}_+ \leq \frac{1}{R_0} \\ \Rightarrow \quad & T_+^{up*}(\mathbb{M}_+, \mathbb{R}_+) > t + T_o + \alpha T_m^{max} + T_r^{max}. \end{aligned}$$

Then if $t + T_o + \alpha T_m^{max} + T_r^{max} < D$, there exists at least one scaling-up solution such that the

inequality $T_+^{up*}(\mathbb{M}_+, \mathbb{R}_+) < D$ holds. Furthermore,

$$\begin{aligned} & t < D - T_o - T_m^{max} - T_r^{max} \\ \Rightarrow \quad & t < D - T_o - \alpha T_m^{max} - T_r^{max} \\ \Leftrightarrow \quad & t + T_o + \alpha T_m^{max} + T_r^{max} < D \end{aligned}$$

Consequently, if the scaling is done before the time $t = D - T_o - T_m^{max} - T_r^{max}$, there exists at least one scaling-up solution to guarantee the QoS, i.e., $T_+^{up*}(\mathbb{M}_+, \mathbb{R}_+) < D$. ∎

Remark 6 *The condition* (3.25) *is a sufficient condition for QoS guarantee, not a necessary one.*

The above remark indicates that even if the condition is not met, the QoS may still be met. But if the condition is met, the QoS is certainly guaranteed.

3.4.3 Theoretical Analysis of the Scale-down Issue

The *QoS-guaranteed Scaling-down* theorem gives sufficient conditions for resource scaling-down for the QoS guarantee. It is used to help the resource provisioning framework determine how much resource needs to be scaled down. The theorem is formally presented below.

Theorem 3 (QoS-guaranteed Scaling-down) *The scaling-down solution* $(\mathbb{M}_-, \mathbb{R}_-)$ *is a QoS-guaranteed scaling-down solution if the condition* (3.26) *is satisfied when* $m_t > 0$ *or the condition* (3.27) *is satisfied when* $m_t = 0$ *and* $r_t > 0$.

$$\begin{aligned} & a^{'}\mathbb{M}_- + b^{'}\mathbb{R}_- \leq c^{'}, \\ & \frac{1}{M_0} \leq \mathbb{M}_- \leq 1, \\ & \frac{1}{R_0} \leq \mathbb{R}_- \leq 1 \end{aligned} \tag{3.26}$$

$$\mathbb{M}_- \geq \frac{1}{M_0},\ \frac{1}{R_0} \leq \mathbb{R}_- \leq \min\left\{\frac{d^{'}}{b^{'}}, 1\right\}, \tag{3.27}$$

where $a^{'}$, $b^{'}$, $c^{'}$, and $d^{'}$ are respectively expressed by the following equations:

$$a^{'} = (m_t - 1)T_m^{avg}, \tag{3.28}$$

$$b^{'} = (r_t - 1)T_r^{avg}, \tag{3.29}$$

$$c^{'} = D - t - T_m^{max} + T_m^{avg} - T_r^{max} + \gamma T_r^{avg}, \tag{3.30}$$

$$d^{'} = D - t - T_r^{max} + \gamma T_r^{avg}, \tag{3.31}$$

where γ is a binary number, and $\gamma = 0$ if $m_t > 0$.

Proof When $T_-^{up*}(\mathbb{M}_-, \mathbb{R}_-) \leq D$ holds, the inequality $T_-(\mathbb{M}_-, \mathbb{R}_-) \leq D$ becomes true, implying that the QoS is guaranteed. If $m_t > 0$, no reduce task starts, thus $r_t > 0$. Then it follows from Equations (3.16), (3.17) and (3.18).

$$\begin{aligned} & T_-^{up*}(\mathbb{M}_-, \mathbb{R}_-) \leq D \\ \Leftrightarrow \quad & t + a^{'}\mathbb{M}_- + T_m^{max} - T_m^{avg} + b^{'}\mathbb{R}_- + T_r^{max} - \gamma T_r^{avg} \leq D \\ \Leftrightarrow \quad & a^{'}\mathbb{M}_- + b^{'}\mathbb{R}_- \leq c^{'}. \end{aligned}$$

Furthermore, we cannot remove all map workers/workers when the map/reduce phase is not over, so

$$\begin{aligned} & 0 \leq M_- \leq M_0 - 1,\ 0 \leq R_+ \leq R_0 - 1 \\ \Leftrightarrow \quad & \frac{1}{M_0} \leq \mathbb{M}_- \leq 1,\ \frac{1}{R_0} \leq \mathbb{R}_- \leq 1. \end{aligned}$$

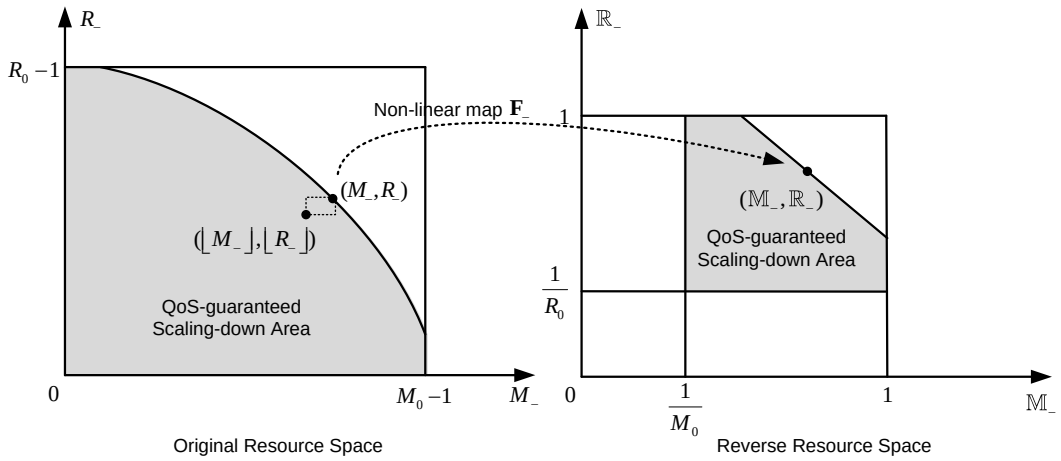
If $m_t = 0$ and $r_t > 0$, we have

$$\begin{aligned} & T_-^{up*}(\mathbb{M}_-, \mathbb{R}_-) \leq D \\ \Leftrightarrow \quad & t + b^{'}\mathbb{R}_- + T_r^{max} - \gamma T_r^{avg} \leq D \\ \Leftrightarrow \quad & \mathbb{R}_- \leq \frac{d^{'}}{b^{'}}. \end{aligned}$$

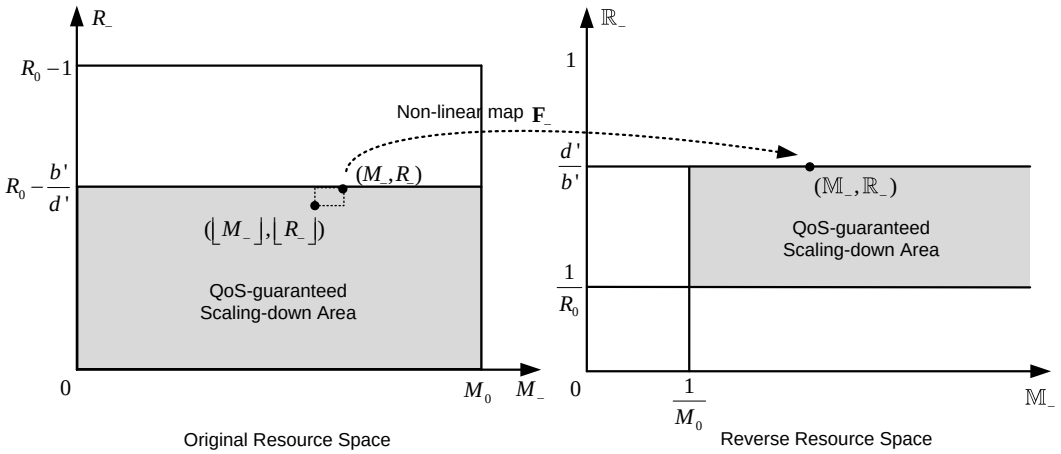
In addition, it is allowed to remove all map workers but at most $R_0 - 1$ reduce workers when the map phase is over while the reduce phase is still under way. Therefore,

$$0 \leq M_- \leq M_0,\ 0 \leq R_- \leq R_0 - 1 \quad \Leftrightarrow \quad \mathbb{M}_- \geq \frac{1}{M_0},\ \frac{1}{R_0} \leq \mathbb{R}_- \leq 1.$$

If $m_t = 0$ and $r_t = 0$, the computation finishes. As we investigate only the resource scaling for a running MapReduce computation, this situation is not considered. Consequently, the QoS-guaranteed scaling-down conditions (3.26) and (3.27) are derived. ∎



(a) $m_t > 0$



(b) $m_t = 0$ and $r_t > 0$

Figure 3.3: QoS-guaranteed scaling-down areas

Remark 7 *The QoS-guaranteed scaling-down condition* (3.26) *is a sufficient condition, not a necessary one.*

The above remark indicates that when the condition is met, the QoS is guaranteed. However, there might be some occasions when the condition is not met but the QoS can still be met.

Remark 8 *Both cases of the QoS-guaranteed scaling-down condition* (3.26) *are expressed in linear inequalities. This significantly simplifies our mathematical operations in theoretical analysis.*

The following remark explains the area in Figure 3.3.

Remark 9 *Each of the two cases of the QoS-guaranteed scaling-down condition* (3.26) *indicates an area that is referred to as the QoS-guaranteed Scaling-down Area.*

The following remark describes how a point in the reverse resource space is re-mapped to a solution in the original resource space, as shown in Figure 3.3.

Remark 10 *Any point* $(\mathbb{M}_-, \mathbb{R}_-)$ *in the QoS-guaranteed scaling-down area is re-mapped to a QoS-guaranteed scaling-down solution* $(\lfloor M_- \rfloor, \lfloor R_- \rfloor)$ *in the original resource space by the non-linear map,* F_-*, where* $M_- = M_0 - 1/\mathbb{M}_-$ *and* $R_- = R_0 - 1/\mathbb{R}_-$.

The *Resource Scaling-down Condition* corollary is derived from the above theorem. It can be used to help the resource provisioning framework judge whether the resource is allowed to be scaled down.

Corollary 2 (Resource Scaling-down Condition) *Resource scaling-down is allowed if the condition* (3.32) *or* (3.33) *is satisfied when* $m_t > 0$*, or the condition* (3.34) *is satisfied when* $m_t = 0$ *and* $r_t > 0$.

$$\frac{a^{'}}{M_0} + \frac{b^{'}}{R_0 - 1} \leq c,\ \frac{a^{'}}{b^{'}} \geq \frac{M_0(M_0 - 1)}{R_0(R_0 - 1)}. \tag{3.32}$$

$$\frac{a^{'}}{M_0 - 1} + \frac{b^{'}}{R_0} \leq c,\ \frac{a^{'}}{b^{'}} < \frac{M_0(M_0 - 1)}{R_0(R_0 - 1)}. \tag{3.33}$$

$$\frac{d^{'}}{b^{'}} \geq \frac{1}{R_0}. \tag{3.34}$$

Proof When $m_t > 0$, as shown in Figure 3.4, the QoS-guaranteed scaling-down area is surrounded by the lines $\mathbb{M}_+ = 1/M_0$, $\mathbb{R}_+ = 1/R_0$, and l_1 (or l_2, l_3). The lines l_1, l_2, and l_3 are expressed by

$$l_1 : a^{'}\mathbb{M}_+ + b^{'}\mathbb{R}_+ = c^{'},\ where\ \frac{a^{'}}{b^{'}} > \frac{M_0(M_0 - 1)}{R_0(R_0 - 1)};$$

$$l_2 : a^{'}\mathbb{M}_+ + b^{'}\mathbb{R}_+ = c^{'},\ where\ \frac{a^{'}}{b^{'}} = \frac{M_0(M_0 - 1)}{R_0(R_0 - 1)};$$

$$l_3 : a^{'}\mathbb{M}_+ + b^{'}\mathbb{R}_+ = c^{'},\ where\ \frac{a^{'}}{b^{'}} < \frac{M_0(M_0 - 1)}{R_0(R_0 - 1)}.$$

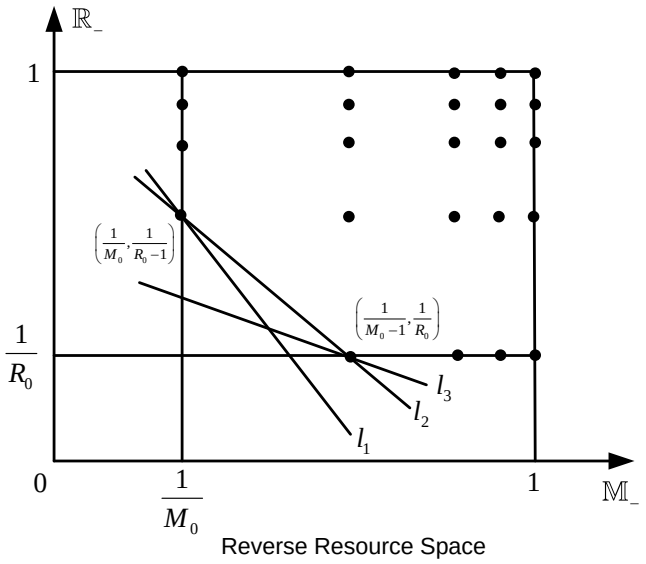


Figure 3.4: Distribution of the scaling-down solutions when $m_t > 0$

Moreover, the points in the figure represent the scaling-down solutions. Notice that the solution $(1/M_0, 1/R_0)$, which is equivalent to $(0, 0)$ in the original resource space, is not a scaling-down solution.

If $a^{'}/b^{'} \geq M_0(M_0-1)/[R_0(R_0-1)]$, we have

$$\begin{aligned} & \frac{a^{'}}{M_0} + \frac{b^{'}}{R_0 - 1} \leq c \\ \Leftrightarrow \quad & \left(\frac{1}{M_0}, \frac{1}{R_0 - 1}\right) \text{ locates in the QoS-guaranteed scaling-down area} \\ \Rightarrow \quad & \text{at least one QoS-guaranteed scaling-down solution exists.} \end{aligned}$$

If $a^{'}/b^{'} < M_0(M_0-1)/[R_0(R_0-1)]$, we obtain

$$\begin{aligned} & \frac{a^{'}}{M_0 - 1} + \frac{b^{'}}{R_0} \leq c \\ \Leftrightarrow \quad & \left(\frac{1}{M_0 - 1}, \frac{1}{R_0}\right) \text{ locates in the QoS-guaranteed scaling-down area} \\ \Rightarrow \quad & \text{at least one QoS-guaranteed scaling-down solution exists.} \end{aligned}$$

In addition, as seen from Figure 3.3(b), when $m_t = 0$ and $r_t > 0$, we can derive that

$$\frac{d^{'}}{b^{'}} \geq \frac{1}{R_0} \Rightarrow \text{at least one QoS-guaranteed scaling-down solution exists.}$$

Consequently, resource scaling-down is allowed when at least one QoS-guaranteed scaling-down solution exists. ∎

3.5 Applications of the Theoretical Results

This section discusses how theoretical results derived in Section 5 can be applied to the resource scaling of cloud-based MapReduce computations to meet their deadlines; and it validates theoretical results by experiments. For this purpose, an experimental environment was created to emulate Infrastructure as a Service (IaaS); cloud-based MapReduce computations use the resources (VMs) from the emulated IaaS to execute their computation tasks.

Two typical cloud-based MapReduce computations were selected from a popular MapReduce benchmark suite *HiBench* [Huang et al., 2010] in our experiments. One was *TeraSort*, a standard MapReduce sort benchmark; another was *WordCount*, an application counting the number of occurrences of each word in a text file. For both the MapReduce computations, the input sizes were 2 GB; the numbers of map and reduce tasks were 32 and 15, respectively.

3.5.1 Applications of the Theorems for Resource Scaling-up

The theorems for resource scaling-up include the QoS-guaranteed Scaling-up theorem and the Latest Intervention Time theorem. They are applied in cloud-based MapReduce computations as follows. Cloud-based MapReduce continuously monitors the resource provisioning for cloud-based MapReduce computations at runtime. When it finds that the amount of the resource provisioning for a cloud-based MapReduce computation meets the condition (3.19) presented in the QoS-guaranteed Scaling-up theorem, cloud-based MapReduce determines a time point of resource scaling-up, which satisfies the condition (3.25) given by the Latest Intervention Time theorem. Meanwhile, cloud-based MapReduce determines how many map workers and reduce workers need to be added to that cloud-based MapReduce computation in the following way. Firstly, cloud-based MapReduce chooses a point in the reverse resource space from the QoS-guaranteed scaling-up area given by the QoS-guaranteed Scaling-up theorem. Then cloud-based MapReduce re-maps the point in the reverse resource space to a point in the original resource space through the re-mapping method presented in Remark 5. The coordinates of the point in the original resource space give the number of map workers and the number of reduce workers

that need to be added to that cloud-based MapReduce computation. Finally, the map and reduce workers are added to that cloud-based MapReduce computation at that time point.

The following experiment was conducted to verify the QoS-guaranteed Scaling-up theorem and the Latest Intervention Time theorem. Firstly, both the TeraSort and WordCount computations were given with the same deadlines of 240 seconds, and were initially provisioned with three map workers and three reduce workers. Without resource scaling-up, the TeraSort and WordCount computation times were 300 seconds and 292 seconds, respectively, and therefore both required resource scaling-up.

During the runtime of the TeraSort/WordCount computation, a time point satisfying the condition (3.25) given by the Latest Intervention Time theorem was chosen for resource scaling-up. For the TeraSort computation, the time point of resource scaling-up was chosen from 40, 80, 120, 160, and 192 seconds, while for the WordCount computation, it was chosen from 40, 80, 120, 160, and 203 seconds. The time points 192 and 203 seconds were the theoretical latest intervention times for the TeraSort and WordCount computations, respectively.

At the time point of resource scaling-up, a point in the reverse resource space was randomly chosen from the QoS-guaranteed scaling-up area presented in the QoS-guaranteed Scaling-up theorem, and then the respective numbers of map workers and reduce workers to be added to the TeraSort/WordCount computation were derived from that point in the reverse resource space in the same way as explained above. The map and reduce workers were added to the TeraSort/WordCount computation at that time point, and the actual computation time of the TeraSort/WordCount computation was recorded when it finished. The above experiment was repeated 10 times at each of the time points of resource scaling-up; consequently, there were 50 results of the TeraSort/WordCount computation times in total.

Figure 3.5 and Figure 3.6 show the 50 results of the TeraSort/WordCount computation times after resource scaling-up. In this figure, the X-axis indicates the time points of resource scaling-up, while the Y-axis indicates the TeraSort/WordCount computation times after scaling up different amounts of resource. The boxes in the figures indicate the ranges of the variations in the computation times after scaling up different amounts of resource. The line under the box indicates the shortest computation time after resource scaling-up while the line or cross above the box indicates the longest computation time after resource scaling-up. As seen from the figure, at each of the time points of resource scaling-up, none of the WordCount and TeraSort

computations missed their deadlines, which verified the condition (3.19) presented in the QoS-guaranteed Scaling-up theorem to be sufficient. Also, when the resource scaling-up is done before/at the latest intervention time, both the WordCount and TeraSort computations met the deadline, which verified the condition (3.25) given by the Latest Intervention Time theorem to be sufficient.

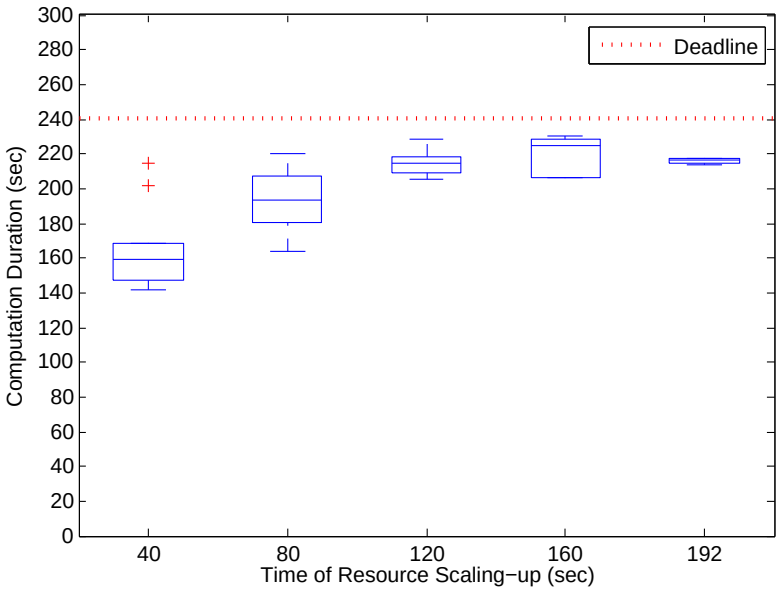


Figure 3.5: Distributions of the computation times of the WordCount computations after resource scaling-up

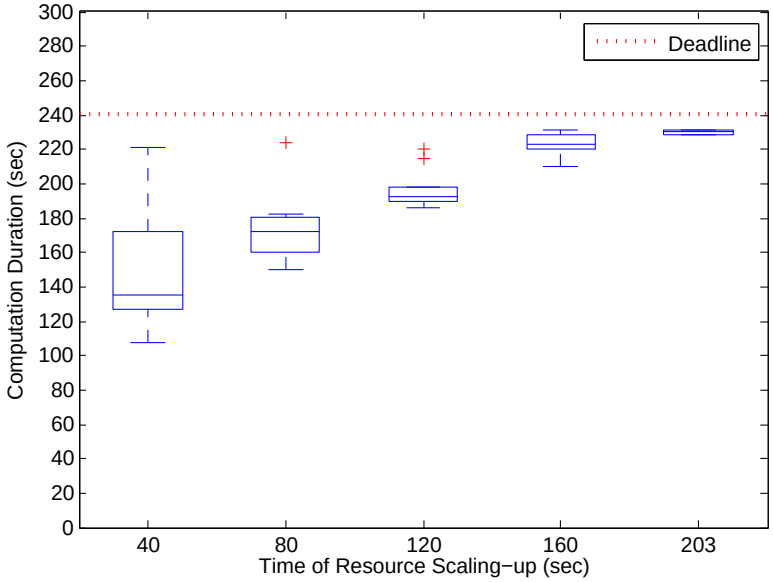


Figure 3.6: Distributions of the computation times of the TeraSort computations after resource scaling-up

3.5.2 Applications of the Theorem for Resource Scaling-down

The theorem for resource scaling-down is the QoS-guaranteed Scaling-down theorem. It is applied in cloud-based MapReduce computations as follows. Cloud-based MapReduce continuously monitors the resource provisioning for cloud-based MapReduce computations at runtime. When it finds that the amount of the resource provisioning for a cloud-based MapReduce computation meets the condition (3.32) presented in the Resource Scaling-down corollary, cloud-based MapReduce determines a time point of resource scaling-down. Meanwhile, cloud-based MapReduce determines how many map workers and reduce workers need to be removed from that cloud-based MapReduce computation in the following way. Firstly, cloud-based MapReduce chooses a point in the reverse resource space from the QoS-guaranteed scaling-down area given by the QoS-guaranteed Scaling-down theorem. Then cloud-based MapReduce re-maps the point in the reverse resource space to a point in the original resource space through the re-mapping method presented in Remark 10. The coordinates of the point in the original resource space give the number of map workers and the number of reduce workers that need to be removed from that cloud-based MapReduce computation. Finally, the map and reduce workers are removed from that cloud-based MapReduce computation at that time point.

The following experiment was conducted to verify the QoS-guaranteed Scaling-down theorem. Firstly, both the TeraSort and WordCount computations were given with the same deadlines of 300 seconds, and were initially provisioned with eight map workers and eight reduce workers. Without resource scaling-down, the TeraSort and WordCount computation times were 115 seconds and 109 seconds, respectively, and therefore both required resource scaling-down.

Resource scaling-down was done at one of the time points, 15, 30, 45, 60 and 75 seconds. At the time point of resource scaling-down, a point in the reverse resource space was randomly chosen from the QoS-guaranteed scaling-down area presented in the QoS-guaranteed Scaling-down theorem, and then the respective numbers of map workers and reduce workers to be removed were derived from that point in the reverse resource space in the same way as explained above. The map and reduce workers were removed from the TeraSort/WordCount computation at that time point, and the actual computation time of the TeraSort/WordCount computation was recorded when it finished. The above experiment was repeated 10 times at each of the time points of resource scaling-down; consequently, there were 50 results of the TeraSort/WordCount

computation times in total.

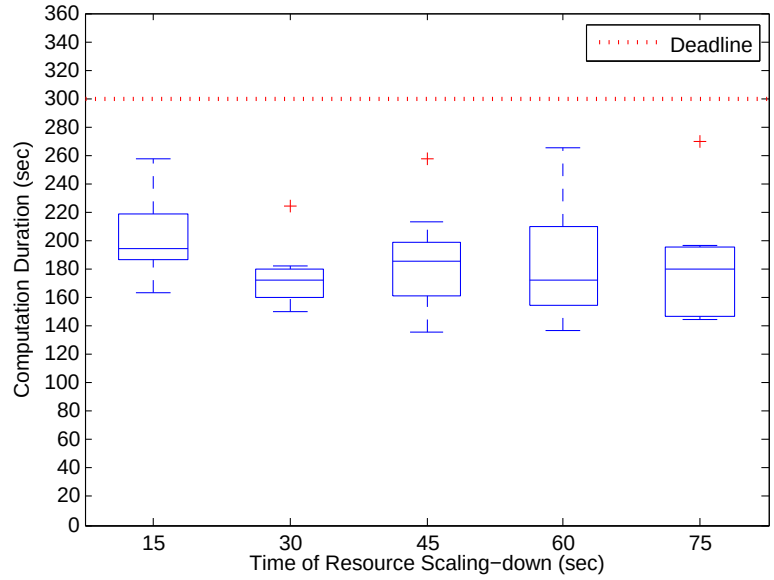


Figure 3.7: Distributions of the computation times of the WordCount computations after resource scaling-down

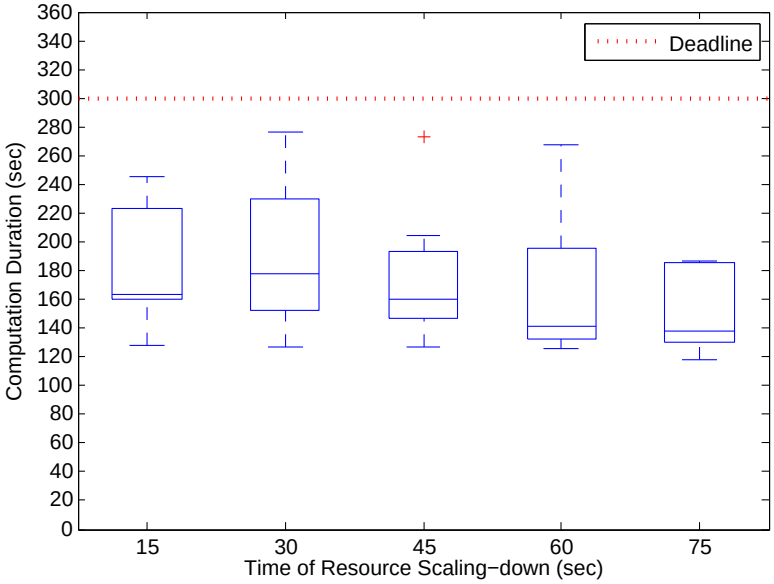


Figure 3.8: Distributions of the computation times of the TeraSort computations after resource scaling-down

Figure 3.7 and Figure 3.8 illustrate the 50 results of the TeraSort/WordCount computation times after resource scaling-down. In this figure, the X-axis indicates the time points of resource scaling-down, while the Y-axis indicates the TeraSort/WordCount computation times after scaling down different amounts of resource. The boxes in the figure indicate the ranges of the variations in the computation times after scaling down different amounts of resource. The line

under the box indicates the shortest computation time after resource scaling-down while the line or cross above the box indicates the longest computation time after resource scaling-down. As shown in the figure, at each of the time points of resource scaling-down, both the WordCount and TeraSort computations met their deadline, verifying the condition (3.26) presented in the QoS-guaranteed Scaling-down theorem to be sufficient.

3.6 Summary of Chapter

The theoretical study of QoS-guaranteed resource provisioning for cloud-based MapReduce has been investigated in this chapter for cloud-based MapReduce computation. The problem has been re-defined in the reverse resource space, which is a Euclidean Plane transformed from the original Euclidean Plane through a non-linear transformation. In the reverse space, the problem has been significantly simplified. Sufficient resource scaling conditions and the latest intervention time have been established for QoS-guaranteed scaling-up/down. These theoretical results provide a theoretical foundation for guaranteeing the QoS of cloud-based MapReduce with minimum operational costs. The research outcome of this chapter has been published in the paper of Xu et al. [2016].

Chapter 4

Cloud-based MapReduce Placement

The study of the Cloud-based MapReduce placement (MRP) problem is to answer an important research question about how to place the MapReduce computations on VMs with minimum operational costs. The results from the study of the MRP problem will be used to guide the resource provisioning framework to conduct resource allocation.

The chapter is organized as follows. Section 4.1 gives the introduction of the study of the MRP problem. Section 4.2 formulates the MRP problem as a constrained combinatorial optimization problem. Section 4.3 designs the new MapReduce placement algorithm. Section 4.4 evaluates the effectiveness and efficiency of the algorithm by comparing it with several baseline algorithms. Section 4.5 summarizes this chapter.

4.1 Introduction

MapReduce is originally proposed for parallel computation in a cluster which consists of a set of connected computers. The objectives of the cluster-based MapReduce computations usually focus on minimizing execution time [Herodotou and Babu, 2011, Lin et al., 2010b, Wolf et al., 2010, Zaharia et al., 2008] or maximizing cluster utilization [Polo et al., 2011, Wang et al., 2011]. However, in cloud-based MapReduce, the most important objective is to guarantee the Quality of Service (QoS) of cloud-based MapReduce computations with the minimum cost of using virtual machines (VMs). To guarantee the QoS, the required number of workers must be placed on a selected set of VMs such that the resource requirements of each worker must be met and the total cost of using the VMs is minimum. This is the so-called *MapReduce Placement*

(MRP) problem in cloud-based MapReduce.

The approaches to the MRP problem can be classified into two categories: *homogeneous MapReduce placement optimization* and *heterogeneous MapReduce placement optimization*. The homogeneous MapReduce placement optimization approaches usually place the workers on a set of homogeneous VMs and place the same number of workers on each of the VMs. Since this category of approaches is easy to implement, most of the existing approaches to the MRP problem belong to this category [AbdelBaky et al., 2012, Cardosa et al., 2012, Chen et al., 2014b, Herodotou et al., 2011, Hwang and Kim, 2012, Lama and Zhou, 2012, Palanisamy et al., 2014, Tian and Chen, 2011]. Very recently, a heterogeneous MapReduce placement optimization approach was proposed [Xu and Tang, 2014a]. That can utilize heterogeneous VMs and place different numbers of workers on different VMs. It showed that the proposed heterogeneous MapReduce placement optimization approach is more cost-effective than those homogeneous MapReduce placement optimization approaches. However, the proposed approach did not reuse the VMs used by old MapReduce computations. This thesis presents a new heterogeneous MapReduce placement optimization approach that considers not only new VMs of various types, but also the spare CPU and memory capacities of existing VMs.

Here is a simple example to illustrate how heterogeneous MapReduce placement optimization potentially outperforms homogeneous MapReduce placement optimization. In this example, it is assumed that there are only two types of VMs, *small VMs* and *large VMs*, and that the capacity of a VM is measured by the number of CPUs (or cores). Each of the small VM contains three CPUs and its price is $4/hour; each of the large VMs contains six CPUs and its price is $6/hour. Let us say there is one new MapReduce computation which requires four identical workers, each of which needs two CPUs. If we adopt homogeneous MapReduce placement, we would have to use either four small VMs or two large VMs. The total costs of using the VMS would be $16/hour and $12/hour, respectively. However, if we adopt heterogeneous MapReduce placement, we would need only one small VMs and one large VMs, and the total cost of using the VMs is only $10/hour. It can be seen from this simple example that heterogeneous MapReduce placement has the potential to cut the total cost of cloud-based MapReduce computations. In this example, we did not reuse the VMS used by existing MapReduce computations. If there is an existing MapReduce computation when a new MapReduce computation comes, and the existing MapReduce computation is using a VM which has two spare CPUs, then we use only one new large VM to accommodate the new MapReduce

computation, and the total extra cost would be only $6/hour.

In this chapter, a new heterogeneous approach to the MRP problem in cloud-based MapReduce computations will be proposed. It has more potential to reuse VMs than existing heterogeneous approaches, and therefore can further reduce the total operational cost of cloud-based MapReduce computations. The MRP problem will be formulated into a constrained combinatorial optimization problem and will also be proven as an NP-complete problem. A new constructive algorithm for the constrained combinatorial optimization problem will be designed and will also be evaluated by experiments.

4.2 Problem Formulation

The cloud-based MapReduce, built on top of a set of VMs of various types rented from a public cloud, can perform multiple MapReduce computations concurrently. New MapReduce computations may arrive and existing MapReduce computations may finish and go at any time. In order to minimize the ongoing operational cost of cloud-based MapReduce, we should minimize the operational cost of cloud-based MapReduce at any time.

In order to minimize its operational cost, cloud-based MapReduce may use a number of different types of VMs which have different capacities and prices. Thus, a fundamental problem is to find which types of VMs should be rented, the numbers of instances of each selected VM type and the placement of the map and reduce workers (workers) on those rented VMs, such that the total cost of renting the VMs is minimum while guaranteeing the QoS of cloud-based MapReduce computation platform at any time.

It is assumed that there are n new MapReduce computations arriving and n' existing MapReduce computations when the MapReduce placement is carried out. In order to guarantee the QoS of the i^{th} new MapReduce computation ($1 \leq i \leq n$), at least t_i^M map workers and t_i^R reduce workers need to be provided for the map/reduce tasks of the new MapReduce computation, and need to be placed on VMs where their resource requirements are met. The map and reduce workers provided for the new MapReduce computations are respectively expressed by two tuples, $< M_i^{CPU}, M_i^{Mem} >$ and $< R_i^{CPU}, R_i^{Mem} >$, where M_i^{CPU} and M_i^{Mem} are the CPU and memory requirements of the map tasks of the i^{th} new MapReduce computation ($1 \leq i \leq n$), and R_i^{CPU} and R_i^{Mem} are the CPU and memory requirements of the reduce tasks of the i^{th} new

MapReduce computation.

The map and reduce workers provided for the existing MapReduce computations are respectively expressed by two tuples, $< {M'}_i^{CPU}, {M'}_i^{Mem} >$ and $< {R'}_i^{CPU}, {R'}_i^{Mem} >$, where ${M'}_i^{CPU}$ and ${M'}_i^{Mem}$ are the CPU and memory requirements of the map tasks of the i^{th} existing MapReduce computation ($1 \leq i \leq n'$), and ${R'}_i^{CPU}$ and ${R'}_i^{Mem}$ are the CPU and memory requirements of the reduce tasks of the i^{th} existing MapReduce computation.

All the map/reduce workers of the MapReduce computations are required to be placed on VMs. The VMs include a set of new VMs, denoted by $\mathbb{V}$, rented from the public cloud, and a set of existing VMs, denoted by $\mathbb{V}'$, which are being used by existing MapReduce computations and which have some spare resources. The new VMs can be classified into m types in terms of their resource capacities and prices and the existing VMs can be classified into m' types in terms of their spare resource capacities; and $\mathbb{V} = \bigcup_{j=1}^{m} \mathbb{V}_j$, $\mathbb{V}' = \bigcup_{j=1}^{m'} {\mathbb{V}'}_j$, where $\mathbb{V}_j$ is a multiset of new VMs of type j and ${\mathbb{V}'}_j$ is a multiset of existing VMs of type j.

In addition, let V_k be an instance of the VMs to be used in the MapReduce placement, where $V_k \in \mathbb{V} \cup \mathbb{V}'$, $1 \leq k \leq |\mathbb{V}| + |\mathbb{V}'|$, and V_k has a CPU capacity, v_k^{CPU} and a memory capacity v_k^{Mem}. Let

$$V_k^s =< x_{k1}^M, x_{k2}^M, \cdots, x_{kn}^M, x_{k1}^R, x_{k2}^R, \cdots, x_{kn}^R >$$

be the assignment of the map and reduce workers of the new MapReduce computations to $V_k \in \mathbb{V} \cup \mathbb{V}'$, where x_{ki}^M and x_{ki}^R are the numbers of the map and reduce workers of the i^{th} new MapReduce computation assigned to V_k and $1 \leq i \leq n$; and let

$$V_k^{s'} =< c_{k1}^M, c_{k2}^M, \cdots, c_{kn'}^M, c_{k1}^R, c_{k2}^R, \cdots, c_{kn'}^R >$$

be the assignment of the map and reduce workers of the existing MapReduce computations to $V_k \in \mathbb{V}'$, where c_{ki}^M and c_{ki}^R are the numbers of the map and reduce workers of the i^{th} existing MapReduce computation assigned to V_k.

Given the entire set of existing VMs that have spare resources, $\mathbb{V}'$, and the placement of the map and reduce workers of the n' existing MapReduce computations on the VMs in $\mathbb{V}'$, the MRP problem is to find a set of new VMs, $\mathbb{V}$, and placements of the map and reduce workers of the n new MapReduce computations on all the new and existing VMs, such that the total cost

of those new VMs is minimal, that is,

$$\min \sum_{j=1}^{m} p_j \cdot |\mathbb{V}_j| \tag{4.1}$$

subject to

$$\sum_{k=1}^{|\mathbb{V}|+|\mathbb{V}'|} x_{ki}^{M} = t_i^{M}, 1 \leq i \leq n \tag{4.2}$$

$$\sum_{k=1}^{|\mathbb{V}|+|\mathbb{V}'|} x_{ki}^{R} = t_i^{R}, 1 \leq i \leq n \tag{4.3}$$

$$\sum_{i=1}^{n} (x_{ki}^{M} \cdot M_i^{CPU} + x_{ki}^{R} \cdot R_i^{CPU}) \leq V_k^{CPU}, \forall V_k \in \mathbb{V}_j \tag{4.4}$$

$$\sum_{i=1}^{n} (x_{ki}^{M} \cdot M_i^{Mem} + x_{ki}^{R} \cdot R_i^{Mem}) \leq V_k^{Mem}, \forall V_k \in \mathbb{V}_j \tag{4.5}$$

$$\sum_{i=1}^{n} (x_{ki}^{M} \cdot M_i^{CPU} + x_{ki}^{R} \cdot R_i^{CPU}) + \sum_{i=1}^{n'} (c_{ki}^{M} \cdot {M'}_i^{CPU} + c_{ki}^{R} \cdot {R'}_i^{CPU}) \leq V_k^{CPU}, \forall V_k \in {\mathbb{V}'}_j \tag{4.6}$$

$$\sum_{i=1}^{n} (x_{ki}^{M} \cdot M_i^{Mem} + x_{ki}^{R} \cdot R_i^{Mem}) + \sum_{i=1}^{n'} (c_{ki}^{M} {M'}_i^{Mem} + c_{ki}^{R} {R'}_i^{Mem}) \leq V_k^{Mem}, \forall V_k \in {\mathbb{V}'}_j \tag{4.7}$$

In this problem formulation, p_j is the price of the j^{th} type of VM. The constraints (4.2) and (4.3) ensure the required numbers of map and reduce workers of all the new MapReduce computations are placed on the VMs; the constraints (4.4) and (4.5) make sure the total CPU and memory requirements of the map/reduce workers on a new VM do not exceed its CPU and memory capacities; the constraints (4.6) and (4.7) guarantee the total CPU and memory

requirements of the map/reduce workers of the new MapReduce computations and the existing MapReduce computations on an existing VM do not exceed its CPU and memory capacities.

The MRP problem is NP-complete, and the proof is presented by the following theorem.

Theorem 4 *The MRP problem is NP-complete.*

Proof *The MRP problem is a special case of the classical bin packing problem [Dyckhoff, 1990], where the workers are objects and the VMs are containers and where the volume of an object (worker) is its CPU requirement and the volume of a container (VM) is the VM's CPU capacity. Let the memory requirement of all the objects be r^M, which is a constant, and the memory capacity of a container (VM) be $N * r^M$, where N is the total number of objects (workers). Then the packing is constrained only by the VM's CPU capacity, and not the VM's memory capacity. In addition, let the cost of each VM be the same, amounting to one dollar. Thus, in this special case, the MRP problem can be transformed into the classical bin packing problem: given a set of objects (workers), how to pack these objects into the minimum number of containers (VMs). Since the classical bin packing problem is NP-complete [Dyckhoff, 1990], the MRP problem is also NP-complete.* ∎

4.3 Algorithm for the MRP Problem

This section describes the algorithm for the MRP problem when new MapReduce computations arrive and when completed MapReduce computations leave, respectively. The algorithm is an approximation algorithm. The MRP problem is NP-complete and its size is usually large; hence it is not feasible to adopt an optimum algorithm to solve the problem as that would lead to an explosion in its search space. However, the approximation algorithm searches only the space where the optimum solutions possibly allocate, so it can solve the problems of large sizes, but without scarifying too much performance on solution quality.

The approximation algorithm is a basically constructive algorithm, which is broken down into two consecutive procedures: *placement pattern generation* and *MRP problem solution building*. The first procedure is used to generates a small set of placement patterns; the second procedure is used to find a combination of the placement patterns that form a solution to the MRP problem with a minimum total cost for using VMs.

A *placement pattern* for a VM of one type is a combination of workers of various types that can be placed on the VM of that type, satisfying the capacity constraints of the VM of that type. A placement pattern is taken as feasible if the total CPU and memory requirements of those workers placed on that type of VM do not exceed the CPU and memory requirements of the VM of that type, respectively. The details about the placement patterns generation procedure and the MRP problem solution building procedure are discussed in the following subsections.

4.3.1 Placement Pattern Generation Procedure

The basic idea behind the placement pattern generation procedure is to use an FFD-based algorithm to generate a set of placement patterns for each type of VM, where the VM is a container and there are many instances of the container, and the workers are objects that need to be put into the multiple containers. Algorithm 1 describes a procedure that generates a set of placement patterns for a particular type of VM.

Algorithm 1 Generating a set of placement patterns for j^{th} type of VM

1: $\mathbb{W}_j = \emptyset$, $\overline{\mathbb{S}}_j = \emptyset$;
2: **for** $i = 1$ to $|\mathbb{W}|$ **do**
3: **if** the CPU/memory requirement of the worker $w_i \in \mathbb{W}$ does not exceed the CPU/memory capacity of a VM of the j^{th} type **then**
4: $\mathbb{W}_j = \mathbb{W}_j \cup \{w_i\}$;
5: **end if**
6: **end for**
7: **for** $k = 1$ to q **do**
8: **for** $i = 1$ to $|\mathbb{W}_j|$ **do**
9: $S_i = \emptyset$;
10: **end for**
11: randomly generate a sequence of the workers in $\mathbb{W}_j$, L;
12: **while** $L \neq \emptyset$ **do**
13: get the first worker w from L;
14: put w into the first VM container that can accommodate it;
15: remove w from L;
16: **end while**
17: **for** $i = 1$ to $|\mathbb{W}_j|$ **do**
18: **if** $S_i \neq \emptyset$; **then**
19: $\overline{\mathbb{S}}_j = \overline{\mathbb{S}}_j \cup S_i$
20: **end if**
21: **end for**
22: **end for**
23: output $\overline{\mathbb{S}}_j$;

The input of Algorithm 1 is the entire multiset of workers, $\mathbb{W}$, which are needed to be placed

on multiple instances of the j^{th} type of VM. The output of the algorithm is a set of placement patterns for the j^{th} type of VM, $\overline{\mathbb{S}}_j$.

In order to make the algorithm more efficient, it first of all sorts out those workers which cannot be put into any of the containers because their 'size' is bigger than that of any container, which is done by checking if their resource requirements exceed the capacity of the container in steps 2-6 of the algorithm. Then the algorithm iterates q times (steps 7-22) of a variant of the FFD algorithm, namely *random FFD algorithm* (steps 8-16) in which the order of the objects (workers) is randomly generated, rather than in descending order by their 'size'. The reason behind that is that we wanted the procedure to generate different placement patterns in each of the iterations. The total number of containers used in the random FFD algorithm is $|\mathbb{W}_j|$, which is enough to accommodate all the objects (workers). Thus, after the packing process of the random FFD algorithm, there could be some containers which are empty. Thus, we need to get rid of those empty containers (steps 17-21). Each of the non-empty containers, S_i, gives a placement pattern for the j^{th} type of VM, and all the placement patterns generated in the q iterations are stored in $\overline{\mathbb{S}}_j$.

The placement pattern generation procedure is described in Algorithm 2. The input is the entire multiset of the workers needed to be placed, $\mathbb{W}$, and the output is a set of placement patterns for all m types of VMs, $\overline{\mathbb{S}}$.

Algorithm 2 iterates $m + m'$ times (steps 2-5), where m is the total number of types of VMs and m' is the total number of types of existing VMs. It should be noted that we categorize the existing VMs with the same spare CPU and memory capacities into the same type. In each iteration, Algorithm 2 invokes Algorithm 1 to generate a set of placement patterns for one type of VM, $\overline{\mathbb{S}}_j$ (step 3), and then merges those placement patterns stored in $\overline{\mathbb{S}}_j$ into $\overline{\mathbb{S}}$ (step 4). Finally, it outputs $\overline{\mathbb{S}}$.

Algorithm 2 Placement pattern generation

1: $\overline{\mathbb{S}} = \emptyset$;
2: **for** $j = 1$ to $m + m'$ **do**
3: use $Algorithm\ 1$ to generate a set of placement patterns for the j^{th} type of VM, $\overline{\mathbb{S}}_j$;
4: $\overline{\mathbb{S}} = \overline{\mathbb{S}} \cup \overline{\mathbb{S}}_j$;
5: **end for**
6: output $\overline{\mathbb{S}}$;

4.3.2 MRP Problem Solution Building Procedure

After using the above placement pattern generation procedure to find a set of feasible placement patterns for all types of VMs, the MRP problem solution building procedure is used to find the best combination of the placement patterns in $\overline{\mathbb{S}}$ to form a solution to the MRP problem.

From the computational point of view, the MRP problem solution building problem is a constrained combinatorial optimization problem. Considering that the total number of feasible placement patterns is not huge, however, we transform the MRP problem solution building problem into a Mixed Integer Programming (MIP) [Wolsey, 2008] problem as follows:

A placement pattern can be expressed by an N-tuple $s_j^k =< x_{jk}^1, x_{jk}^2, \cdots, x_{jk}^i, \cdots, x_{jk}^N >$, where s_j^k is the k^{th} placement pattern of the j^{th} type of VM, x_{jk}^i is the number of workers of the i^{th} type used in the placement pattern, and N is the total number of different types of workers. It should be noted that the workers with the same CPU and memory requirements are categorized into the same type. The objective of the MIP problem is

$$\min Z = \sum_{j=1}^{m} \sum_{k=1}^{|\overline{\mathbb{S}}_j|} p_j \cdot y_j^k \tag{4.8}$$

subject to

$$\sum_{j=1}^{m+m'} \sum_{k=1}^{|\overline{\mathbb{S}}_j|} x_{jk}^i \cdot y_j^k \geq |\mathbb{W}_i|, 1 \leq i \leq N \tag{4.9}$$

$$\sum_{k=1}^{|\overline{\mathbb{S}}_j|} y_j^k \leq N_j, m < j \leq m + m' \tag{4.10}$$

$$y_j^k \geq 0, 1 \leq k \leq |\overline{\mathbb{S}}_j|, 1 \leq j \leq m + m' \tag{4.11}$$

In Equation (4.8), Z is the total cost of all the VMs needed in the MapReduce placement, y_j^k is the decision variable representing the number of the placement pattern, s_j^k, used in the MapReduce placement, p_j is the price of the VM of the j^{th} type, $|\overline{\mathbb{S}}_j|$, denotes the total number of the placement patterns for the j^{th} VM type, which is generated in the placement pattern generation procedure. The constraint (4.9) ensures the required number of the workers of every type involved in the MapReduce computations are assigned to one of the VMs. The

constraint (4.10) makes sure the number of each type of existing VMs used in the solution does not exceed its available number, where N_j is the maximum available number of one type of existing VMs. The constraint (4.11) ensures all variables must be non-negative integers.

It should be noted that the total number of workers in the MIP solution could be more than the total number of workers required to be placed in the MRP problem because of the relaxed constraint (4.9). Therefore, we need to remove those redundant workers from the MIP solution before the MIP solution can be used for the MRP problem.

4.4 Evaluation

The evaluation of our new algorithm is done through two experiments. The first experiment is to test the performance of our new constructive algorithm (NCA). In the experiment, we compare NCA with three baseline algorithms in terms of the cost of the MapReduce placements generated by the algorithms for a set of test instances of various characteristics.

One of the baseline algorithms is the most popular algorithm for HOMOgeneous MapReduce placement (HOMO) [Palanisamy et al., 2014]. HOMO selects a suitable type of VM among multiple types of VM and then assigns the same number of workers to multiple instances of the selected type of VM. A second baseline algorithm is an FFD-based MapReduce placement algorithm (FFD-based). The FFD-based algorithm picks workers in a decreasing order by their resource requirements and places them in a first-fit fashion. Details about this algorithm can be found in the work of Kang and Park [2003]. A third baseline algorithm is the original constructive algorithm (OCA) presented in the work of Xu and Tang [2014a]. Both the FFD-based algorithm and NCA reuse those spare resources on existing VMs whereas HOMO and OCA do not. All the algorithms except for HOMO are designed for heterogeneous MapReduce placement.

The second experiment is to test the scalability of NCA, which is done by observing how the computation time of NCA increases when the size of the test problems increases.

Both of the experiments were conducted on a laptop with an Intel Core i7-3520M CPU (2.90 GHz) and 8 GBs of RAM. All the VMs used in the experiments were generated by VMware Workstation 10.0.0 [VMware, 2015], and were deployed on 12 HP workstations (32 Intel Xeon 2.40 GHz CPUs and 320 GB memory) interconnected via a Gigabit Ethernet network. Hadoop

0.20.2 [Hadoop, 2015] was used to run the MapReduce benchmarks and Ganglia [Ganglia, 2015] was used to monitor the resource consumption during runtime. All of the algorithms used in the experiments were implemented in C#. The solver for the MIP in the MRP problem solution building procedure is CPLEX (12.5.1.0) [CPLEX, 2015].

4.4.1 Construction of Test Instances

In the evaluation, two benchmarks for MapReduce computations were selected from a popular MapReduce benchmark suite, namely *HiBench* [Huang et al., 2010], and were used to construct a number of test instances of different sizes, each of which was used as a test problem in the experiments. One benchmark was *TeraSort*, a standard MapReduce sort benchmark; another was *WordCount*, an application that counts the number of occurrences of each word in a text file.

Each test instance had three inputs: the number of MapReduce computations, the number of workers in each of the MapReduce computations, and the information about existing VMs. The types of VMS used in the experiments are shown in Table 4.1.

Table 4.1: VM types

VM Type	CPUs (#Cores)	Mem (GB)	Cost ($)
m1 small	1	1.7	0.06
m1 medium	2	3.75	0.12
m1 large	4	7.5	0.24
m1 xlarge	8	14.7	0.48
m2 xlarge	6.5	17.1	0.41
m2 2xlarge	13	34.2	0.82
c1 medium	5	1.7	0.145
c1 xlarge	20	7	0.58

When constructing test instances, the resource requirements of map/reduce workers were compacted by experiments. Table 4.2 shows the resource requirements of the map/reduce workers for these two benchmarks with different input sizes. The resource requirements shown in the table are the average results of 10 runs.

Using the information shown in Table 4.2, the following methods were used to construct more and large-size test instances. It was assumed that the CPU and memory requirements of map workers were uniformly distributed in the interval $[a, b]$, where a was the observed minimum amount of resource requirement and b was the observed maximum amount of resource

Table 4.2: Resource requirements of the workers with different input sizes

	Input	Mapper		Reducer	
	Size (GB)	CPUs (#Cores)	Mem (GB)	CPUs (#Cores)	Mem (GB)
TeraSort	2	[1.5,1.8]	[0.1,0.2]	1.12	0.9
	4	[1.5,1.8]	[0.1,0.2]	1.32	1.3
	6	[1.5,1.8]	[0.1,0.2]	1.4	1.65
	8	[1.5,1.8]	[0.1,0.2]	1.52	1.8
	10	[1.5,1.8]	[0.1,0.2]	1.68	2
WordCount	4	[1.7,1.9]	[0.3,0.4]	0.68	0.15
	8	[1.7,1.9]	[0.3,0.4]	0.85	0.4
	12	[1.7,1.9]	[0.3,0.4]	1.08	0.59
	16	[1.7,1.9]	[0.3,0.4]	1.2	0.7
	20	[1.7,1.9]	[0.3,0.4]	1.29	0.85

requirement. Thus, the CPU requirement for a test instance with a fixed input size was randomly picked up between a and b.

The CPU and memory requirements of reduce workers were generated in different way. It was observed that the requirements for CPU and memory were in proportion to the input size of the MapReduce computation. Thus, to generate the CPU and memory requirements for reduce workers, the following four linear regressions was applied to find the relationship between the CPU/memory requirement and the MapReduce computation input size:

$$y_c^{ts} = 0.066x + 1.012 \tag{4.12}$$

$$y_m^{ts} = 0.135x + 0.72 \tag{4.13}$$

$$y_c^{wc} = 0.0393x + 0.549 \tag{4.14}$$

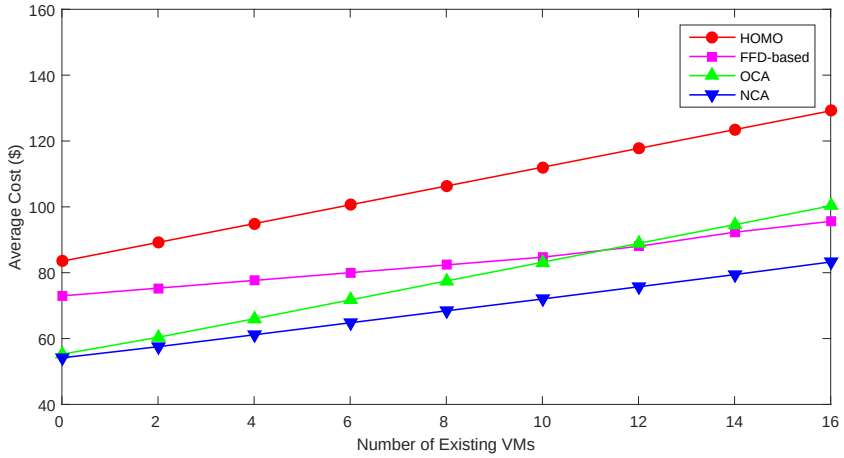
$$y_m^{wc} = 0.0425x + 0.028 \tag{4.15}$$

where x is the input size, y_c^{ts} (y_m^{ts}) indicates the requirement for CPU (memory) of the reduce workers for TeraSort, y_c^{wc} (y_m^{wc}) denotes the requirement for CPU (memory) of the reduce workers for WordCount. Given any input size, x, uniformly distributed in the interval $[10-120]$, we calculated the resource requirements for reduce workers using these four equations.

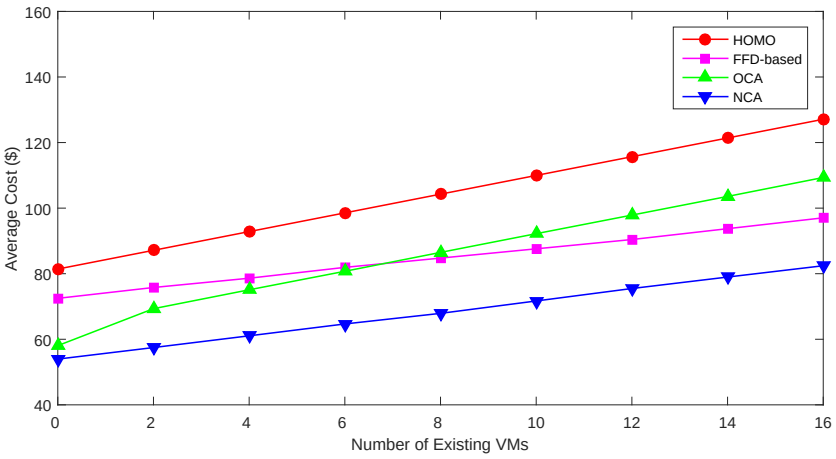
4.4.2 Experiments and Results

In the experiments, we used HOMO, the FFD-based algorithm, OCA and NCA to solve each of the test instances. Because of the stochastic nature of OCA and NCA, we used them 20 times to

solve each of the test instances and used the averages of the 20 runs to compare with the other two algorithms. The maximum time for solving the MIP problem in the MRP problem solution building phase of OCA and NCA was set to 30 seconds, following the suggestion in the work of Haouari and Serairi [2009]. The parameter q used in Algorithm 2 was fixed to 10 after a number of trials.



(a) TeraSort



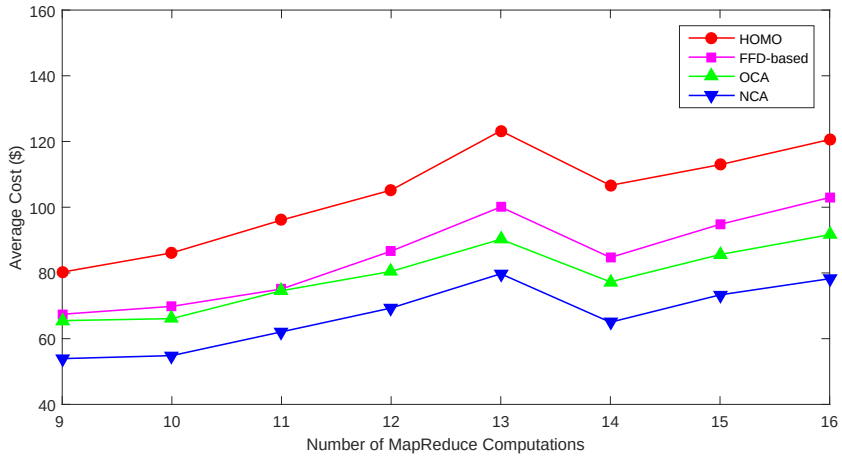
(b) WordCount

Figure 4.1: Comparison of the four algorithms on the cost of using VMs when the number of existing VMs varied

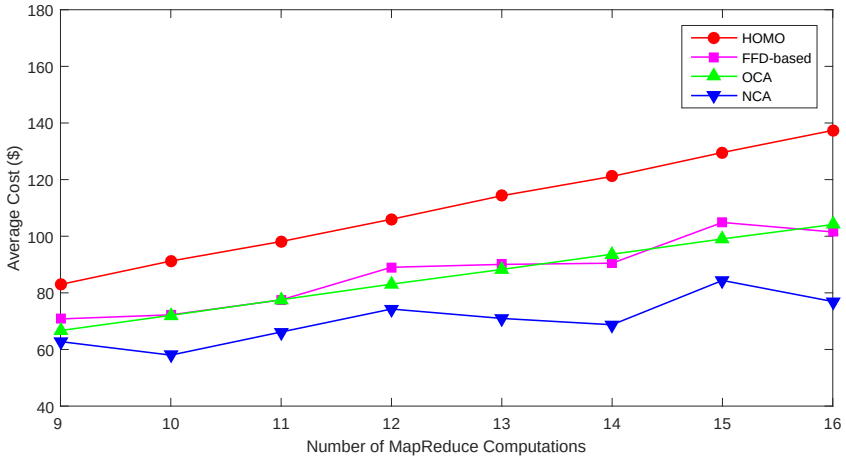
Figure 4.1 shows how the costs of using VMs vary when the four algorithms are used to solve the test instances of TeraSort and WordCount, respectively. In the experiments, the number of existing VMs varied from 0 to 18, the number of worker types was fixed at 24, and the number of workers of each type was fixed at 20. It was assumed in the experiments that the remaining resource on each existing VM was 50 percent of the total resource.

As seen from the figure, when the number of existing VMs was zero, or there was no existing

VMs, the cost of the MapReduce placement generated by NCA was 35.1 percent less than that of HOMO, 25.8 percent less than that of the FFD-based algorithm and 2.0 percent less than that of OCA for those test instances of TeraSort, and 33.7 percent less than that of HOMO, 25.6 percent less than that of the FFD-based algorithm and 7.1 percent less than that of OCA for those test instances of WordCount. It can be also seen from Figure 1 that when there were existing VMs, the cost of the MapReduce placement generated by NCA was $35.5 - 35.7$ percent less than that of HOMO, $12.9 - 23.6$ percent less than that of the FFD-based algorithm and $4.7 - 17.0$ percent less than that of OCA for those test instances of TeraSort, and $34.0 - 35.1$ percent less than that of HOMO, $15.1 - 24.2$ percent less than that of the FFD-based algorithm and $17.1 - 24.6$ percent less than that of OCA for those test instances of WordCount.



(a) TeraSort

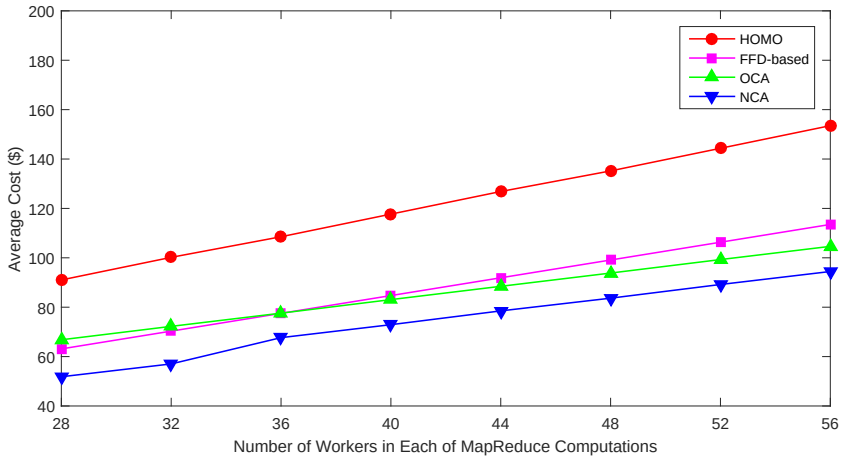


(b) WordCount

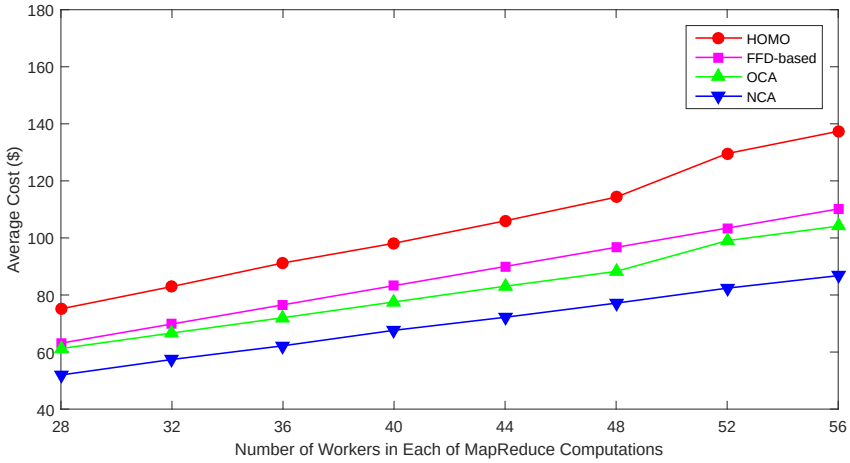
Figure 4.2: Comparison of the four algorithms on the cost of using VMs when the number of MapReduce computations varied

Figure 4.2 compares the costs of the MapReduce placement solutions generated by the

four algorithms for TeraSort and WordCount, respectively, when the number of MapReduce computations varies from 9 to 16. In the experiments, the number of workers in each of the MapReduce computations was fixed at 40, the number of the existing VMs of each type was fixed at 10, and the remaining resource on each existing VM was 50 percent of the total resource. For the test instances of TeraSort, the cost of the MapReduce placement generated by NCA was $32.8 - 39.1$ percent less than that of HOMO, $17.5 - 27.0$ percent less than that FFD-based algorithm, and $13.2 - 21.4$ percent less than that of OCA. For the test instances of WordCount, the cost of the MapReduce placement generated by NCA was $24.3 - 44.0$ percent less than that of HOMO, $12.8 - 31.9$ percent less than that of the FFD-based algorithm, and $6.2 - 36.2$ percent less than that of OCA.



(a) TeraSort

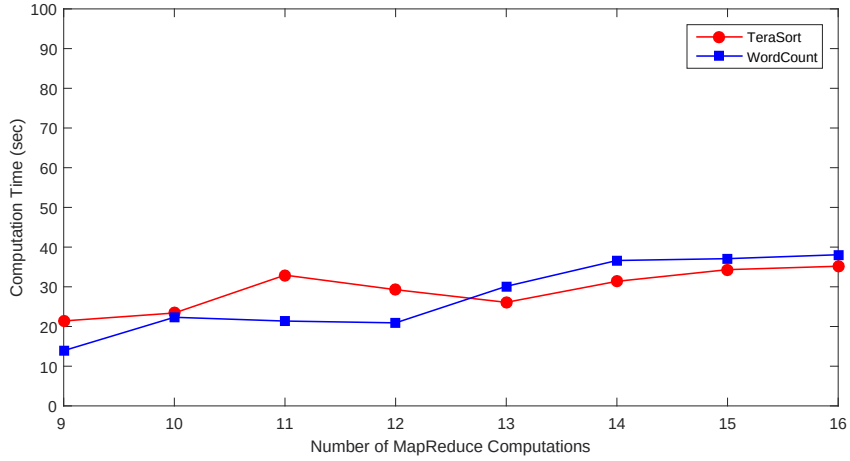


(b) WordCount

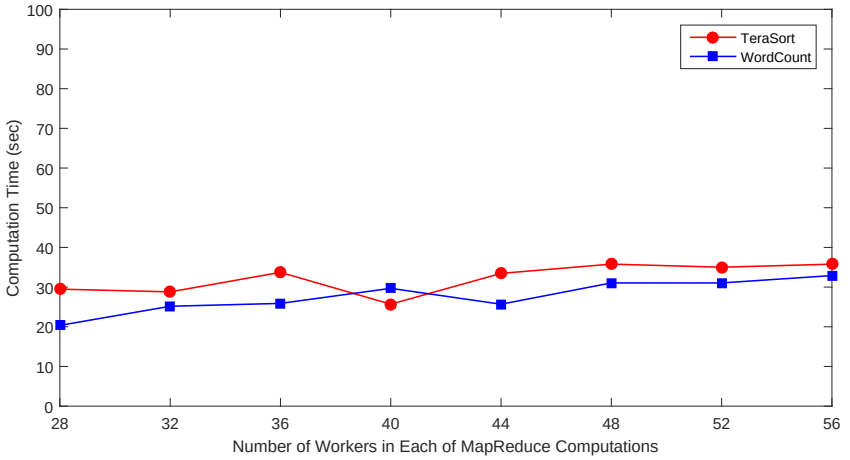
Figure 4.3: Comparison of the four algorithms on the cost of using VMs when the number of workers in each of the MapReduce computations varied

Figure 4.3 presents the comparison of the cost of the MapReduce placement solutions

generated by the four algorithms for TeraSort and WordCount, when the number of workers in each of the MapReduce computations varies from 28 to 56. In the experiments, the number of MapReduce computations was fixed at 12, the number of the existing VMs of each type was fixed at 10, and the remaining resource on each existing VM was 50 percent of the total resource. For the test instances of TeraSort, the cost of the MapReduce placement generated by NCA was $37.6 - 43.1$ percent less than that of HOMO, $12.7 - 18.4$ percent less than that of the FFD-based algorithm, and $10.8 - 28.9$ percent less than that of OCA. For the test instances of WordCount, the cost of the MapReduce placement generated by NCA was $30.8 - 36.8$ percent less than that of HOMO, $17.7 - 21.2$ percent less than that of the FFD-based algorithm, and $12.6 - 16.8$ percent less than that of OCA.



(a)



(b)

Figure 4.4: Scalability of NCA

Figure 4.4 displays the experiments on the scalability of NCA. In the experiments, the number of the existing VMs of each type was fixed at 10 and the remaining resource on each

existing VM was 50 percent of the total resource. As seen from the figure, the computation time of NCA increased linearly when the number of MapReduce computations increased, and that the computation time of NCA did not change significantly when the number of workers in each of the MapReduce computations varied.

In summary, NCA always had better performance than all the three baseline algorithms for all the tested problems. In addition, the good scalability of NCA was demonstrated.

4.4.3 Discussion

As illustrated by the experimental results, our new algorithm outperformed other three baseline algorithms, including HOMO, the FFD-based algorithm and OCA, on saving the cost of using VMs. Particularly, our new algorithm was better than HOMO and the FFD-based algorithm since it found better combinations of the worker placements that form a solution to the MRP problem. A better combination of worker placements means less cost of using VMs. Thus, the MRP solution found by our new algorithm consumed less cost of using VMs than those by HOMO and the FFD-based algorithm.

Our new algorithm also performed better OCA, as it could reuse the spare resource of existing VMs. That greatly helped our new algorithm used less new VMs than OCA. Thus, using our new algorithm the cost of using VMs was less than that using OCA.

4.5 Summary of Chapter

This chapter has proposed a new algorithm for the cloud-based MapReduce placement (MRP) problem and has evaluated the new algorithm by experiments. The experimental results have demonstrated the effectiveness of the new algorithm as a heterogeneous placement algorithm. The operational cost of cloud-based MapReduce computation platform using the new algorithm was $24.3 - 44.0$ percent lower than that using the most popular homogeneous MapReduce placement algorithm. The experimental results have also shown that the new algorithm is more efficient than another two heterogeneous MapReduce placement algorithms. The operational cost of cloud-based MapReduce computation platform using the new algorithm was $12.7 - 31.9$ percent lower than that using the FFD-based algorithm, and $2.0 - 36.2$ percent lower than that using the heterogeneous MapReduce placement algorithm not considering the spare

resources from the existing MapReduce computations. Finally, the experimental results have demonstrated the good scalability of the new algorithm. The research outcome of this chapter has been published in the paper of Xu et al. [2015].

Chapter 5

Cloud-based MapReduce Consolidation

The study of the Cloud-based MapReduce consolidation (MRC) problem is to answer the research question regarding how to consolidate the remaining MapReduce computations on VMs with minimum operational costs when some MapReduce computations are competing. The results from the study of the MRC problem will be used to guide the resource provisioning framework to conduct MapReduce consolidation.

The chapter is organized as follows. Section 5.1 gives the introduction to the study of the MRC problem. Section 5.2 formulates the MRC problem as a bio-objective optimization problem. Section 5.3 designs the new MapReduce consolidation algorithm. Section 5.4 demonstrates the effectiveness by case studies, and also evaluates the efficiency of the algorithm by comparing it with a baseline algorithm. Section 5.5 summarizes this chapter.

5.1 Introduction

The MRC problem is raised when some MapReduce computations complete and leave, or resource provisioning for some MapReduce computations is scaled down during the runtime of cloud-based MapReduce. At that time, the workers for those MapReduce computations are removed and the resources occupied by the workers are released. The MRC problem is then to consolidate the remaining workers on existing VMs to minimize the cost of using VMs. Meanwhile, the MRC problem considers the migration cost during the MapReduce consolidation. To reduce the system operation, the MRC problem also tries to minimize the migration cost.

The cost of using VMs and the migration cost for cloud-based MapReduce are potentially reduced. Here is a simple example to illustrate how both the cost of using VMs and the migration cost are reduced through addressing the MRC problem. It is assumed there are two VMs, VM A and VM B, both of which have three CPUs (or cores) and 300 MBs and are charged by $1/hour. Each of the VMs has one large worker requiring two CPUs and 200 MBs and one small worker requiring one CPU and 100 MBs. At a time point, a small worker on VM A and a large worker on VM B complete their respective MapReduce computations and are then removed. Without MapReduce consolidation, cloud-based MapReduce could still use two VMs and have to pay for two dollars. However, with MapReduce consolidation, the small and large workers are consolidated on one VM, and cloud-based MapReduce shut down the idle VM and needs to pay for only one dollar so that the cost of using VMs is saved. Meanwhile, there are two consolidation solutions. The first is to move the large worker on VM A to VM B, which has to migrate 200 MBs of data. The second is to move the small worker on VM B to VM A, which migrates only 100 MBs of data. The second solution incurs less migration cost than the first one. Thus, addressing the MRC problem, the cost of using VMs and the migration cost for cloud-based MapReduce are potentially reduced.

In this chapter, the MRC will be formulated into a bio-objective optimization problem and will also be proven as an NP-complete problem. A new constructive algorithm will be designed for the MRC problem, and the new constructive algorithm will be evaluated by experiments. The algorithm firstly narrows the VMs to be consolidated, then finds the promising assignment patterns of MapReduce computations on each type of the VMs to be consolidated, and finally finds an optimum solution through optimizing the combination of those promising assignment patterns. Using these three steps, the algorithm reduces the search scope without sacrificing the possibility of locating optimum solutions.

5.2 Problem Formulation

When the MapReduce computations are completed and leave, the spare resources occupied by those completed MapReduce computations are available. To further save the operational cost of cloud-based MapReduce, the remaining MapReduce computations are re-assigned to existing VMs. After the reassignment, the idle VMs will be shut down to save their operational cost. The migration cost involved in the re-assignment needs to be considered by cloud-based

MapReduce. During the migration from the original placement to the new placement, the procedures and data for a worker, which are stored in the memory of the VM loading that worker, are migrated to the new VM. The cost of transferring those procedures and data is defined as the migration cost.

It is assumed that there are n remaining MapReduce computations when some MapReduce computations are completed. The map and reduce workers of the i^{th} remaining MapReduce computation are respectively expressed by two tuples, $< M_i^{CPU}, M_i^{Mem} >$ and $< R_i^{CPU}, R_i^{Mem} >$, where M_i^{CPU} and M_i^{Mem} are the CPU and memory requirements of the map tasks of the i^{th} remaining MapReduce computation ($1 \leq i \leq n$), and R_i^{CPU} and R_i^{Mem} are the CPU and memory requirements of the reduce tasks of the i^{th} remaining MapReduce computation.

At the same time, there is a multiset of existing VMs, $\mathbb{V}$. Existing VMs are categorized into m types in terms of assignments and prices. In other words, two VMs are of the same type when they have the same assignments and prices. Let $\mathbb{V}_j$ be a multiset of existing VMs of type j, and $\bigcup_{j=1}^{m} \mathbb{V}_j = \mathbb{V}$. After the re-assignment, some VMs may have no map workers or reduce workers on them, and they will be shut down to save cost. Let $\mathbb{V}'$ be the multiset of existing VMs after the re-assignment and $mathcalV' \subseteq \mathbb{V}'$. Let $|\mathbb{V}'_j|$ be the multiset of existing VMs of type j after the re-assignment and $mathcalV'_j \subseteq \mathbb{V}'_j$.

Let $V_k \in \mathbb{V}$ be the k^{th} VM in the set of existing VMs is $\mathbb{V}$, and $1 \leq k \leq |\mathbb{V}|$. The CPU capacity of the VM V_k is denoted by v_k^{CPU} while the memory capacity of the VM V_k is indicated by v_k^{Mem}. Before the re-assignment, the assignment of the map and reduce workers on the VM V_k is denoted by

$$V_k^s =< c_{k1}^M, c_{k2}^M, \cdots, c_{kn}^M, c_{k1}^R, c_{k2}^R, \cdots, c_{kn}^R >$$

where c_{ki}^M and c_{ki}^R are the numbers of the map and reduce workers of the i^{th} remaining MapReduce computation assigned to V_k and $1 \leq i \leq n$. After the re-assignment, the assignment of the map and reduce workers on the VM V_k is changed to

$$V_k^{s'} =< x_{k1}^M, x_{k2}^M, \cdots, x_{kn}^M, x_{k1}^R, x_{k2}^R, \cdots, x_{kn}^R >$$

where x_{ki}^M and x_{ki}^R are the numbers of the map and reduce workers of the i^{th} remaining computation assigned to V_k and $1 \leq i \leq n$;

Cloud-based MapReduce tries to minimize the cost of the VMs for the remaining MapReduce computations,

$$C_v = \sum_{j=1}^{m} |\mathbb{V}'_j| \cdot p_j \tag{5.1}$$

where p_j is the price of the j^{th} type of VMs. Meanwhile, cloud-based MapReduce needs to minimize the migration cost during the re-assignment,

$$C_m = \sum_{k=1}^{|\mathbb{V}|} \sum_{i=1}^{n} \frac{|c_{ki}^M - x_{ki}^M| \cdot M_i^{Mem} + |c_{ki}^R - x_{ki}^R| \cdot R_i^{Mem}}{2} \tag{5.2}$$

The migration cost, C_m, amounts to the total amount of the memories of the workers required to be migrated.

Next, the MRC problem is formulated as follows. Given the entire set of existing VMs, $\mathbb{V}$, and the original placements of the n remaining MapReduce computations on the VMs in $\mathbb{V}$, that is $\{V_k^s | 1 \leq k \leq |\mathbb{V}|\}$, the MRC problem is to find a set of new placements of the n remaining MapReduce computations on the VMs in $\mathbb{V}'$, that is $\{V_k^{s'} | 1 \leq k \leq |\mathbb{V}'|\}$, such that both the cost of using VMs, C_v, and the migration cost, C_m, are minimum, that is,

$$\min C_v \tag{5.3}$$

and

$$\min C_m \tag{5.4}$$

subject to

$$\sum_{k=1}^{|\mathbb{V}'|} x_{ki}^M = \sum_{k=1}^{|\mathbb{V}|} c_{ki}^M, 1 \leq i \leq n \tag{5.5}$$

$$\sum_{k=1}^{|\mathbb{V}'|} x_{ki}^R = \sum_{k=1}^{|\mathbb{V}|} c_{ki}^R, 1 \leq i \leq n \tag{5.6}$$

$$\sum_{i=1}^{n}(x_{ki}^{M} \cdot M_i^{CPU} + x_{ki}^{R} \cdot R_i^{CPU}) \leq V_k^{CPU}, \forall V_k \in \mathbb{V}'_j \tag{5.7}$$

$$\sum_{i=1}^{n}(x_{ki}^{M} \cdot M_i^{Mem} + x_{ki}^{R} \cdot R_i^{Mem}) \leq V_k^{Mem}, \forall V_k \in \mathbb{V}'_j \tag{5.8}$$

In this problem formulation, the bi-objective optimization problem is formulated as a single objective optimization problem through introducing two weights, W_v and W_m, which respectively represent the weights of the cost of using VMs and the migration cost. The constraints (5.5) and (5.6) ensure the numbers of existing map and reduce workers are not changed after the reassignment; the constraints (5.7) and(5.8) make sure the total CPU and memory requirements of the map/reduce workers on a VM do not exceed its CPU and memory capacities.

The MRC problem is NP-complete, and the proof is presented by the following theorem.

Theorem 5 *The MRC problem is NP-complete.*

Proof *The MRC problem is a special case of the classical bin packing problem [Dyckhoff, 1990], where the workers are objects and the VMs are containers and where the volume of an object (worker) is its CPU requirement and the volume of a container (VM) is the VM's CPU capacity. Let the memory requirement of all the objects be r^M, which is a constant, and the memory capacity of a container (VM) be $N * r^M$, where N is the total number of objects (workers). Then the packing is constrained only by the VM's CPU capacity, and not the VM's memory capacity. In addition, let the cost of each VM be the same, amounting to one dollar. Thus, in this special case, the MRC problem can be transformed into the classical bin packing problem: given a set of objects (workers), how to pack these objects into the minimum number of containers (VMs). Since the classical bin packing problem is NP-complete [Dyckhoff, 1990], the MRC problem is also NP-complete.* ∎

5.3 Algorithm for the MRC Problem

This section describes the algorithm for the MRC problem when completed MapReduce computations leave. The algorithm is an approximation algorithm. The MRC problem is NP-complete and its size is is usually huge, it is impossible to adopt an optimum algorithm to solve the

problem considering that would lead to an explosion in its search space. But the approximation algorithm can solve the problems of large sizes without scarifying too much performance on solution quality, as it searches only the space where the optimum solutions possibly allocate.

The approximation algorithm, a basically constructive algorithm, is broken down into three consecutive procedures: *VM selection procedure*, *placement pattern generation* and *MRC solution building*. The first procedure is used to select the VMs where the MapReduce computation consolidation will be conducted. The second procedure is used to generates a small set of placement patterns. The third procedure is used to find a combination of the placement patterns that form an optimum solution to the MRC problem. In particular, the third procedure transforms the bio-objective optimization into the single objective optimization by normalization and introducing weights.

A *placement pattern* for a VM of one type is a combination of workers of various types that can be placed on the VM of that type, satisfying the capacity constraints of the VM of that type. A placement pattern is said to be feasible if the total CPU and memory requirements of those workers that are placed on the VM of that type do not exceed the CPU and memory requirements of the VM of that type, respectively. The details of the placement pattern generation procedure and the MRC problem solution building procedure are discussed in the following subsections.

5.3.1 VM Selection Procedure

The basic idea behind the VM selection procedure is to select a subset of existing VMs to conduct the MapReduce consolidation, such that the migration cost is potentially reduced. In addition, the VM selection procedure prefers to choose those VMs with low resource utilization, potentially reducing the cost of using VMs. The details about the procedure are given in Algorithm 3.

Algorithm 3 VM selection

1: $\mathbb{V}* = \emptyset$;
2: **for** $k = 1$ to $|\mathbb{V}|$ **do**
3: **if** the VM V_k just completed a MapReduce computation **then**
4: $\mathbb{V}* = \mathbb{V}* \cup V_k$;
5: **end if**
6: **end for**
7: output $\mathbb{V}*$;

The input of Algorithm 3 is the entire set of existing VMs, $\mathbb{V}$. The output of Algorithm 5.3.1 is a subset of $\mathbb{V}$, $\mathbb{V}*$, which is the set of VMs. This algorithm iterates all the VMs in $\mathbb{V}$ and selects those VMs which just finished MapReduce computations to conduct the MapReduce consolidation. The basic idea behind the selection is those VMs which just finished MapReduce computations have plenty of spare resources and their resource utilization is low, while the other VMs have few spare resources and high resource utilization, as the MapReduce placement algorithm has well utilized those resources on those VMs.

5.3.2 Placement Pattern Generation Procedure

Having known the VMs where the consolidation needs to be done, $\mathbb{V}*$, the placement pattern generation procedure is to use an FFD-based algorithm to generate a set of placement patterns for existing VMs, and to put the workers on the VMs, $\mathbb{V}*$, into existing VMs.

Algorithm 4 Generating a set of placement patterns for j^{th} type of VM

1: $\mathbb{W}*_j = \emptyset$, $\overline{\mathbb{S}}_j = \emptyset$;
2: **for** $i = 1$ to $|\mathbb{W}|$ **do**
3: **if** the CPU/memory requirement of the worker $w_i \in \mathbb{W}$ does not exceed the CPU/memory capacity of a VM of the j^{th} type **then**
4: $\mathbb{W}_j = \mathbb{W}_j \cup \{w_i\}$;
5: **end if**
6: **end for**
7: **for** $k = 1$ to q **do**
8: **for** $i = 1$ to $|\mathbb{W}_j|$ **do**
9: $S_i = \emptyset$;
10: **end for**
11: randomly generate a sequence of the workers in $\mathbb{W}_j$, L;
12: **while** $L \neq \emptyset$ **do**
13: get the first worker w from L;
14: put w into the first VM container that can accommodate it;
15: remove w from L;
16: **end while**
17: **for** $i = 1$ to $|\mathbb{W}_j|$ **do**
18: **if** $S_i \neq \emptyset$; **then**
19: $\overline{\mathbb{S}}_j = \overline{\mathbb{S}}_j \cup S_i$
20: **end if**
21: **end for**
22: **end for**
23: output $\overline{\mathbb{S}}_j$;

In particular, this algorithm re-assigns only a subset of workers of remaining MapReduce computations, denoted by $\mathbb{W}' \subset \mathbb{W}$, to existing VMs, rather than re-assigning the whole set

of the workers of remaining MapReduce computations, $\mathbb{W}$. And existing VMs do not contain those workers in $\mathbb{W}*$ before the re-assignment. The subset, $\mathbb{W}*$, includes only the workers of the MapReduce computations which share the VMs with the completed MapReduce computations. Such a step greatly reduces the complexity of the system operations and lowers the risk of system instability. It also reduces the search space, speeding up finding the solution.

Algorithm 4 describes a procedure that generates a set of placement patterns for a particular type of VM. The input of Algorithm 4 is the set of the workers on the VMs of the j^{th} type, $\mathbb{W}*_j$. The output of Algorithm 4 is a set of placement patterns for the j^{th} ($1 \leq j \leq m$) type of existing VMs, $\overline{\mathbb{S}}_j$. It should be noted that existing VMs with the same placement patterns and prices are of the same type. The Remaining steps of the Algorithm 4 is almost the same as Algorithm 1, so will not be explained again here.

The placement pattern generation procedure is described in Algorithm 5. The input is the entire multiset of the workers needed to be placed, $\mathbb{W}*$, and the output is a set of placement patterns for all the m types of VMs, $\overline{\mathbb{S}}$. This procedure is almost the same as that for the MRP problem, so it will not be explained again here.

Algorithm 5 Placement pattern generation

1: $\overline{\mathbb{S}} = \emptyset$;
2: **for** $j = 1$ to m **do**
3: use $Algorithm$ 4 to generate a set of placement patterns for the j^{th} type of existing VMs, $\overline{\mathbb{S}}_j$;
4: $\overline{\mathbb{S}} = \overline{\mathbb{S}} \cup \overline{\mathbb{S}}_j$;
5: **end for**
6: output $\overline{\mathbb{S}}$;

5.3.3 MRC problem Solution Building Procedure

After using the placement pattern generation procedure to find a set of feasible placement patterns for all types of VMs, the MRC problem solution building procedure is used to find the best combination of the placement patterns in $\overline{\mathbb{S}}$ to form a solution to the MRC problem.

The MRC problem solution building problem is a constrained combinatorial optimization problem, which can be transformed into a Mixed Integer Programming (MIP) [Wolsey, 2008] problem as follows:

An original placement pattern can be expressed by an N-tuple:

$$s'_j =< c_j^1, c_j^2, \cdots, c_j^i, \cdots, c_j^N >$$

where s'_j is the k^{th} original placement pattern of the j^{th} type of VM, c_j^i is the number of workers of the i^{th} type used in the original placement pattern, and N is the total number of different types of workers. It should be noted that the same type of VMs has the same placement pattern, and the workers with the same CPU and memory requirements are categorized into the same type.

A new placement pattern can be expressed by an N-tuple:

$$s_j^k =< x_{jk}^1, x_{jk}^2, \cdots, x_{jk}^i, \cdots, x_{jk}^N >$$

where s_j^k is the k^{th} placement pattern of the j^{th} type of VM, x_{jk}^i is the number of workers of the i^{th} type used in the placement pattern, and N is the total number of different types of workers.

The MIP transforms these two objectives of the MRC problem into one objective by normalization and introducing weights. The objective function of the MIP problem is given by

$$Z = \left(w_v \cdot \frac{C_v - C_v^{low}}{C_v^{up} - C_v^{low}} + w_m \cdot \frac{C_m - C_m^{low}}{C_m^{up} - C_m^{low}} \right) \tag{5.9}$$

In the objective function, w_v and w_m are the weights of the cost of using VMs and the migration cost, respectively. The notation C_v calculates the cost of using VMs for a solution to the MIP problem, which is expressed by

$$C_v = \sum_{j=1}^{m} \sum_{k=1}^{|\overline{\mathbb{S}}_j|} p_j \cdot y_j^k \tag{5.10}$$

where y_j^k is the decision variable representing the number of the placement pattern, s_j^k, used in the MapReduce placement, p_j is the price of the VM of the j^{th} type, $|\overline{\mathbb{S}}_j|$ denotes the total number of the placement patterns for the j^{th} VM type, which is generated in the placement pattern generation procedure. C_v^{low} and C_v^{up} are the lower and upper bounds of C_v. Obviously,

$$C_v^{low} = 0 \tag{5.11}$$

which means C_v achieves its lower bound when all the VMs are free and can be shut down.

Meanwhile,

$$C_v^{up} = \sum_{j=1}^{m} |\mathbb{V}_j| \cdot p_j \tag{5.12}$$

which means C_v achieves its upper bound when no VMs are shut down and the cost of the VMs for the remaining MapReduce computations is the same as that before the re-assignment.

Also, in the objective function, the notation C_m calculates the migration cost for a solution to the MIP problem, which is expressed by

$$C_m = \sum_{j=1}^{m} \sum_{k=1}^{|\overline{\mathbb{S}}_j|} \left(\sum_{i=1}^{n} \frac{|x_{jk}^i - c_j^i| \cdot r_i}{2} \cdot y_j^k \right) \tag{5.13}$$

The right part of this equation amounts to the total amount of the memories to be migrated. C_m^{low} and C_m^{up} denote the lower and upper bounds of C_m, respectively. Obviously,

$$C_m^{low} = 0 \tag{5.14}$$

which means C_m achieves its lower bound when no workers are migrated. Meanwhile,

$$C_m^{up} = \sum_{k=1}^{|\mathbb{V}|} \sum_{i=1}^{n} \left(c_{ki}^M \cdot M_i^{Mem} + c_{ki}^R \cdot R_i^{Mem} \right) \tag{5.15}$$

which means C_m achieves its upper bound when all the workers are migrated.

Then the MIP is formulated as follows:

$$\min Z \tag{5.16}$$

subject to

$$\sum_{j=1}^{m} \sum_{k=1}^{|\overline{\mathbb{S}}_j|} x_{jk}^i \cdot y_j^k \geq |\mathbb{W}_i|, 1 \leq i \leq N \tag{5.17}$$

$$\sum_{k=1}^{|\overline{\mathbb{S}}_j|} y_j^k \leq N_j, 1 < j \leq m \tag{5.18}$$

$$y_j^k \geq 0, 1 \leq k \leq |\overline{\mathbb{S}}_j|, 1 \leq j \leq m \tag{5.19}$$

In this formulation, the constraint (5.17) ensures the required number of workers of every type involved in the MapReduce computations are assigned to one of the VMs; the constraint (5.18) makes sure the number of each type of existing VMs used in the solution does not exceed its available number, where N_j is the maximum available number of one type of existing VMs; the constraint (5.19) ensures all variables must be non-negative integers.

It should be noted that the total number of workers in the MIP solution could be more than the total number of workers required to be placed in the MRC problem because of the relaxed constraint (5.17). Therefore, we need to remove those redundant workers from the MIP solution before the MIP solution can be used for the MPC problem.

5.4 Evaluation

The evaluation of the new constructive algorithm (NCA) is conducted through two set of experiments. The first is to evaluate the effectiveness of the MapReduce consolidation using NCA. The second is to evaluate the performance of the new algorithm on the cost of using VMs and migration cost, and also to evaluate the performance of NCA on scalability.

In these experiments, two benchmarks for MapReduce computations were selected from a popular MapReduce benchmark suite, namely *HiBench* [Huang et al., 2010], and were used to construct a number of test instances of different sizes, each of which was used as a test problem in the experiments. One benchmark was *TeraSort*, a standard MapReduce sort benchmark; the other was *WordCount*, an application that counts the number of occurrences of each word in a text file.

The experiments were conducted on a laptop with an Intel Core i7-3520M CPU (2.90 GHz) and 8 GBs of RAM. All the VMs used in the experiments were generated by VMware Workstation 10.0.0 [VMware, 2015], and were deployed on 12 HP workstations (32 Intel Xeon 2.40 GHz CPUs and 320 GB memory) interconnected via a Gigabit Ethernet network. Hadoop 0.20.2 [Hadoop, 2015] was used to run the MapReduce benchmarks and Ganglia [Ganglia, 2015] was used to monitor the resource consumption during runtime. All of the algorithms used in the experiments were implemented in C#. The solver for the MIP in the MRC problem solution building procedure is CPLEX (12.5.1.0) [CPLEX, 2015].

A set of test instances of various characteristics were applied in these two sets of experiments. Addressing each of the test instances, the effectiveness and efficiency of NCA were evaluated. The construction process of the test instances is given in the following subsection.

5.4.1 Construction of Test Instances

Each of the generated test instances contained a set of VMs to be consolidated and a placement for each of the VMs. The VMs to be consolidated were those loading the MapReduce computation which was just completed and left; thus they had the spare resources for the remaining MapReduce computations to conduct the consolidation. The number of the VMs to be consolidated was configured in $[20, 40, 60, 80, 100]$. These VMs were randomly chosen from the ones shown in Table 5.1.

Table 5.1: VM types

VM Type	CPUs (#Cores)	Mem (GB)	Cost ($)
m1 small	1	1.7	0.06
m1 medium	2	3.75	0.12
m1 large	4	7.5	0.24
m1 xlarge	8	14.7	0.48
m2 xlarge	6.5	17.1	0.41
m2 2xlarge	13	34.2	0.82
c1 medium	5	1.7	0.145
c1 xlarge	20	7	0.58

There was one placement for each of the VMs to be consolidated, which indicated the assignment of the workers for the remaining MapReduce computations. The workers in the placements were randomly generated from five MapReduce (WordCount or TeraSort) computations. The generated method of the workers was described in Section 4.5.1 of Chapter 4 in detail. The placements were divided into two groups. One group was the empty placements. In these placements, all the workers were for the completed MapReduce computations and were all removed. The percentage of the empty placements was configured to 30 percent based on the observation of the historic runs. Another group was the placements with an amount of spare resources. In these placements, only some workers were removed, and the percentage of the amount of spare resources to the resource capacity was a random figure in $20 - 99$ percent.

For each of the test instances, the consolidation algorithms were used to re-assign the workers involved in the placements to the VMs to be consolidated. The algorithms returned

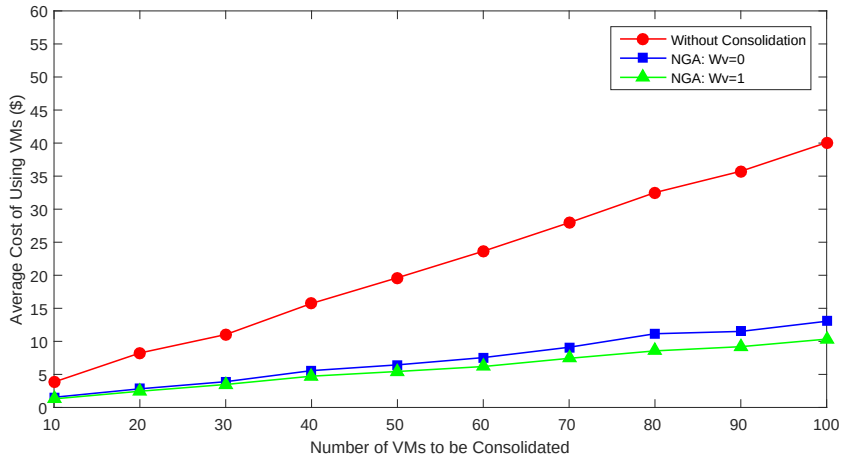
the re-assignment for the same test instance, which was the solution to that test instance.

5.4.2 Effectiveness Evaluation of MapReduce Consolidation

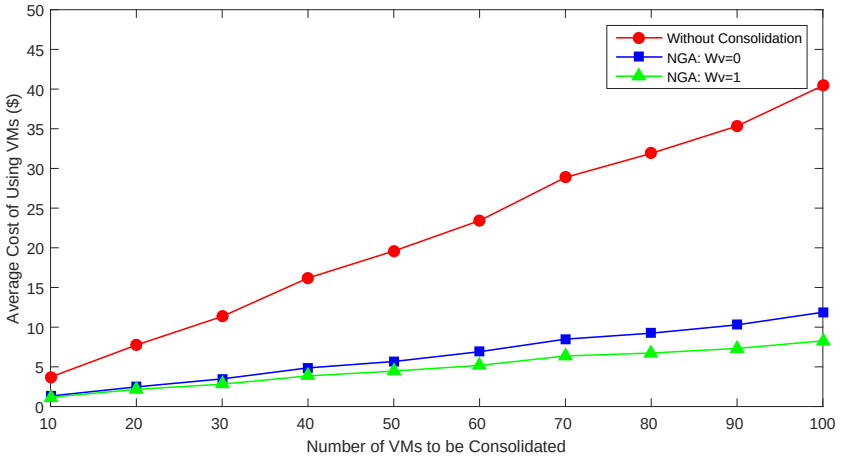
This set of experiments evaluated how much cost was saved by the MapReduce consolidation using NCA. In the experiments, for each of the generated test instances, the cost of using VMs with the MapReduce consolidation was compared with that without the MapReduce consolidation.

Because of the stochastic nature of NCA, we used them 20 times to solve each of the test instances, then used the averages of the 20 runs to compare with the other two algorithms. The maximum time for solving the MIP problem in the MRC problem solution building phase of OCA and NCA was set to 30 seconds, following the suggestion in the work of Haouari and Serairi [2009]. The parameter q used in Algorithm 5 was fixed to 10 after a number of trials.

Figure 5.1 compares the cost of using VMs with the MapReduce consolidation by NCA with that without MapReduce consolidation. In the figure, the notation *NCA:* $W_v = 0$ denoted the weight of the cost of using VMs in NCA was set to 0, which meant the cost of using VMs by using NCA achieved its lower bound; the notation *NCA:* $W_v = 1$ indicated the weight of the cost of using VMs in NCA was configured as 1, which meant the cost of using VMs by using NCA achieved its upper bound. As seen from the figures, the cost of using VMs with MapReduce consolidation was always lower than that without MapReduce consolidation. The lower bounds of the costs of using VMs with MapReduce consolidation were $60.0-68.3$ percent and $64.2 - 70.4$ percent less than the cost of using VMs without MapReduce consolidation for both the TeraSort computations and WordCount computations, respectively, when the number of VMs to be consolidated was increased from 10 to 100. Meanwhile, the upper bounds of the costs of using VMs with MapReduce consolidation were $66.3 - 75.4$ percent and $68.6 - 79.2$ percent less than the cost of using VMs without MapReduce consolidation for both the TeraSort computations and WordCount computations, respectively, when the number of VMs to be consolidated was increased from 10 to 100. In addition, both of the figures showed that the cost of using VMs with MapReduce consolidation was reduced as the number of the VMs increased. Thus, using MapReduce consolidation, the cost of using VMs was saved further, and the savings increased even more when the number of VMs to be consolidated increased.



(a) TeraSort



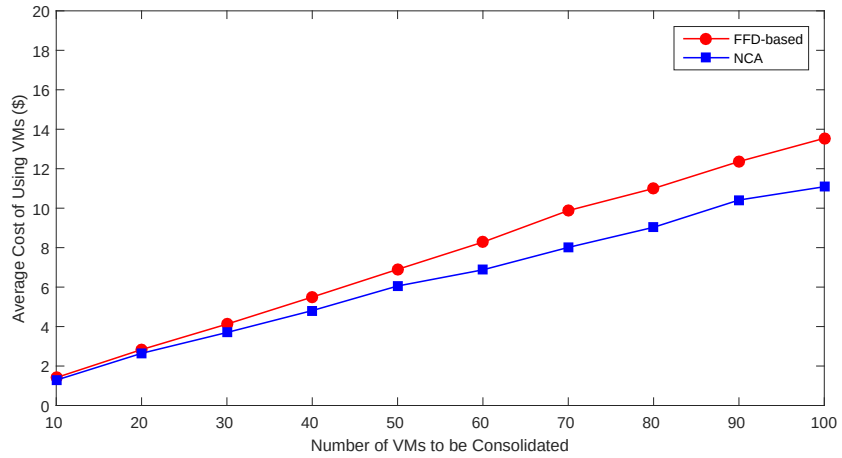
(b) WordCount

Figure 5.1: Comparison of the cost of using VMs with MapReduce consolidation with that without MapReduce consolidation

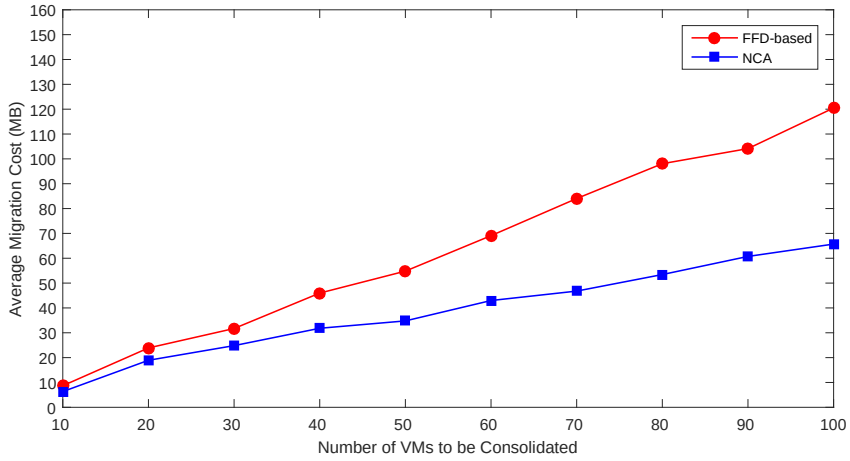
5.4.3 Efficiency Evaluation of Algorithms

This set of experiments evaluated the performance of NCA on the cost of using VMs, the migration cost and the weighted cost which was the value of the objective function 5.9 by comparing it with a baseline algorithm, an FFD-based MapReduce consolidation algorithm (FFD-based). The FFD-based algorithm picked workers in a decreasing order by their resource requirements and placed them in a first-fit fashion. Before solving the test instances, the FFD-based algorithm transformed the bio-objective optimization into the same single objective optimization as that tackled by NCA. Details about this algorithm can be found in the paper [Kang and Park, 2003]. Both of these algorithms set the weight of the cost of using VMs as 0.5.

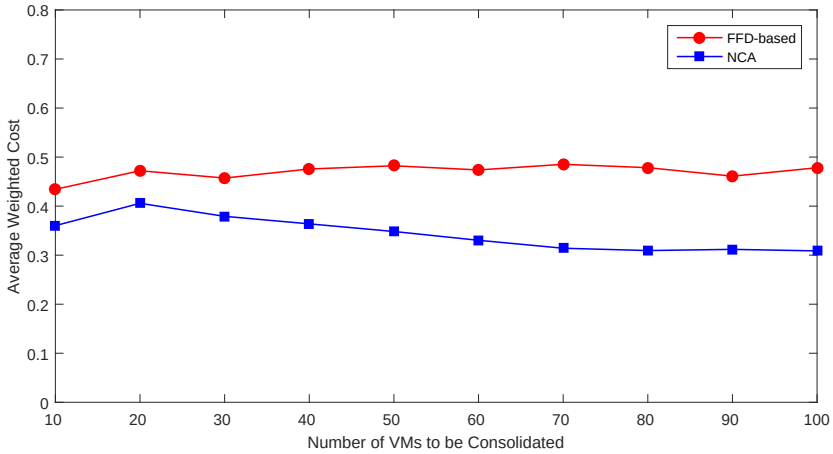
Figure 5.2 shows the results when both of these two algorithms were used to address the



(a) The cost of using VMs



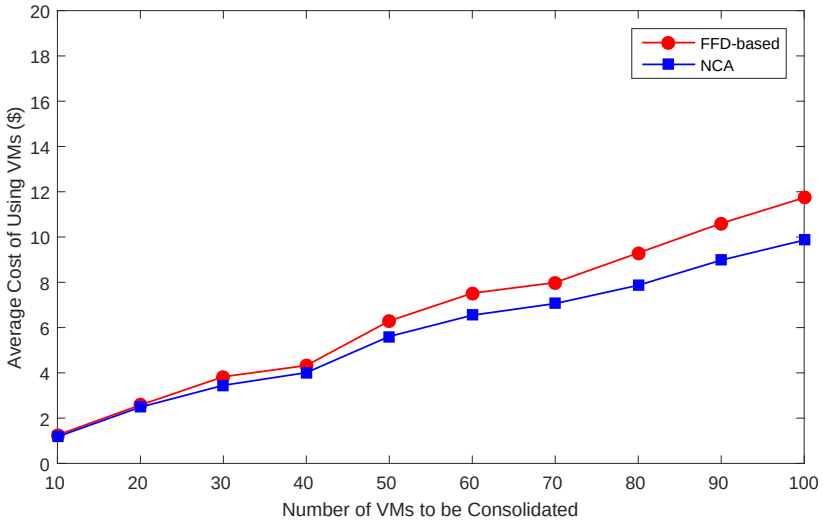
(b) The migration cost



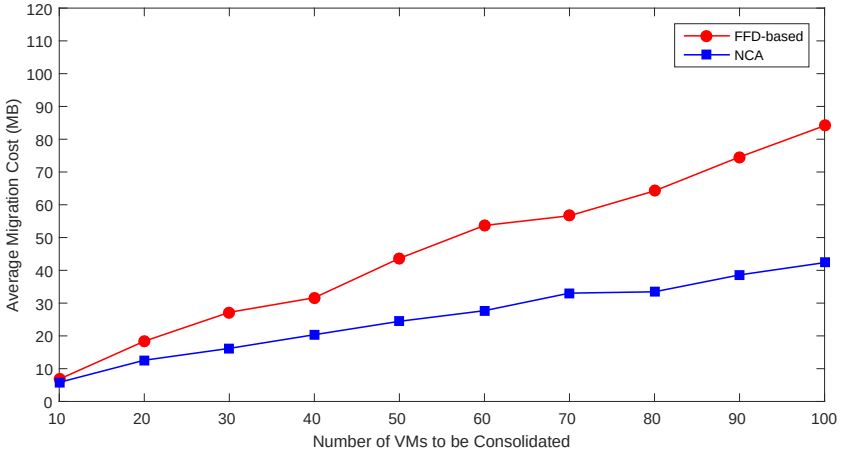
(c) The weighted cost

Figure 5.2: Comparison of NCA and the FFD-based algorithm for the TeraSort computations

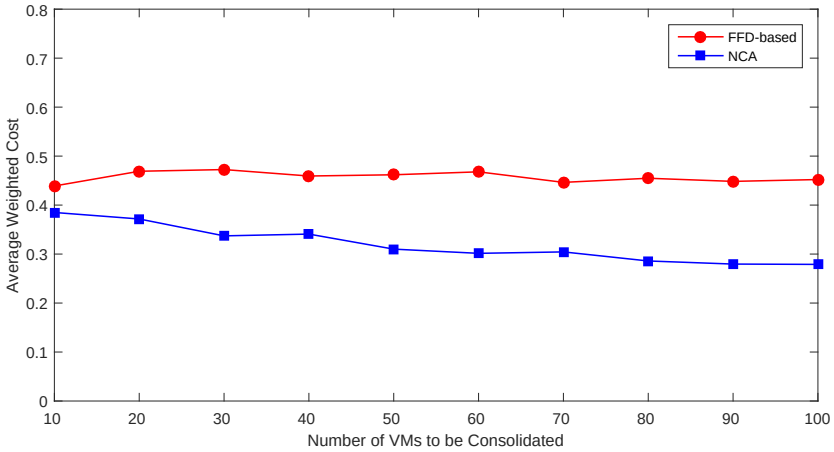
same consolidation problem for the TeraSort computations. As shown in the figures, when the number of VMs to be consolidated increased from 10 to 100, NCA always performed better than the FFD-algorithm. As the number of VMs to be consolidated increased from 10 to 100, the cost



(a) The cost of using VMs



(b) The migration cost



(c) The weighted cost

Figure 5.3: Comparison of NCA and the FFD-based algorithm for the WordCount computations

of using VMs by NCA was $9.0 - 18.1$ percent less than that by the IFFD-based algorithm, the migration cost by NCA was $27.1 - 45.5$ percent less than that by the IFFD-based algorithm, and the weighted cost by NCA was $17.1 - 35.5$ percent less than that by the IFFD-based algorithm.

Figure 5.3 gives the results when both of these two algorithms were used to address the same consolidation problem for the WordCount computations. As shown in the figures, when the number of VMs to be consolidated increased from 10 to 100, NCA always had a better performance than the FFD-algorithm for the WordCount computations. As the number of VMs to be consolidated increased from 10 to 100, the cost of using VMs by NCA was and $6.1 - 16.0$ percent less than that by the IFFD-based algorithm, the migration cost by NCA was $14.3 - 49.7$ percent less than that by the IFFD-based algorithm, and the weighted cost by NCA was $12.3 - 38.3$ percent less than that by the IFFD-based algorithm. In addition, as the number of VMs to be consolidated increased, the weighted cost by NCA gradually decreased, but that by the IFFD-based algorithm increased.

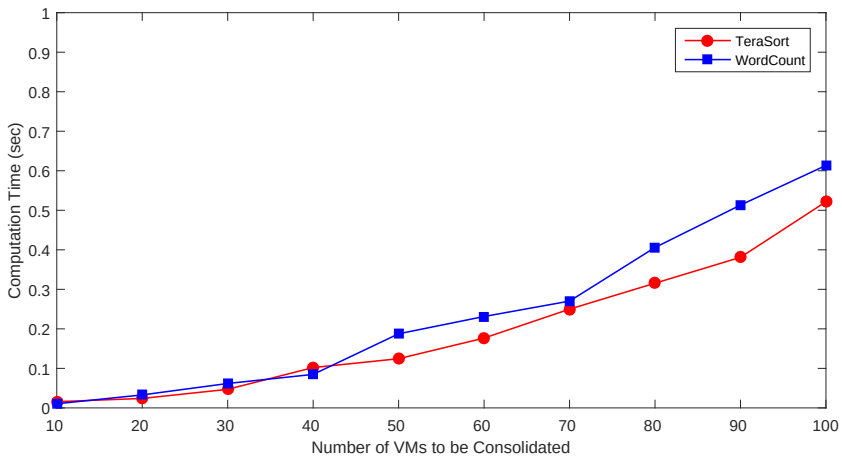


Figure 5.4: Computation times of NCA when the number of VMs to be consolidated changed

This set of experiments also evaluated the scalability of NCA. Figure 5.4 illustrates how the computation times of NCA changed when the number of VMs to be consolidated increased from 10 to 100. As shown in the figure, the computation times of NCA for both the TeraSort computations and WordCount computations increased slowly as the number of VMs to be consolidated increased. Thus, the experimental results showed the scalability of NCA was good.

In summary, NCA always had better performance than the baseline algorithm for all the tested instances on the costing using VMs and the migration cost. In addition, the good scalability of NCA was demonstrated .

5.4.4 Discussion

As illustrated by the experimental results, our new algorithm outperformed the baseline algorithm, the FFD-based algorithm. Compared with the FFD-based algorithm, our new algorithm found more promising combinations of the worker placements by consecutively conducting VM selection, placement pattern generation and solution building. A more promising combination of worker placement means less cost of using VMs. Thus, our new algorithm saved more cost of using VMs than the FFD-based algorithm when conducting MapReduce consolidation.

5.5 Summary of Chapter

This chapter has proposed a new algorithm for the cloud-based MapReduce consolidation (MRC) problem and has evaluated the new algorithm by experiments. The experimental results have demonstrated the effectiveness of the new algorithm. The cost of using VMs with MapReduce consolidation was $60.0 - 79.2$ percent less than that without MapReduce consolidation. The experimental results have also demonstrated the efficiency of the new algorithm. By using the new algorithm, the cost of using VMs was $6.1 - 18.1$ percent less than that using the IFFD-based algorithm, and the migration cost was $14.3 - 49.7$ percent less than that using the IFFD-based algorithm. Finally, the experimental results have demonstrated the good scalability of the new algorithm.

Chapter 6

Development of QoS-guaranteed Resource Provisioning Framework

---

This chapter will present the new resource provisioning framework for cloud-based MapReduce. Section 6.1 presents the introduction. Sections 6.2 and 6.3 characterize and formulate the QoS-guaranteed resource provisioning problem for cloud-based MapReduce. Section 6.4 presents the overall structure of the framework and the trigger mechanism. Section 6.5 introduces the resource provisioning algorithms of the framework. Section 6.6 validates the effectiveness of the framework on guaranteeing the QoS of cloud-based MapReduce by case studies, and also compares the framework with other two popular frameworks on QoS-guarantee and operational cost saving. Section 6.7 gives the summary of the research of this chapter.

6.1 Introduction

The evolution of cluster-based MapReduce to cloud-based MapReduce leads to a new problem namely QoS-guaranteed resource provisioning problem. This problem is how to guarantee the QoS (i.e. hard deadline), while minimizing the cost of renting the VMs to run the MapReduce computations. However, it is a particularly challenging problem in the cloud, a dynamically changing environment. The VMs and network for cloud-based MapReduce computations usually experience performance variability because of the shared underlying infrastructures [Farley et al., 2012]. This performance variability will delay the progress of MapReduce computations and cause QoS violation if the initial resource is under-provisioned, or will result in resource waste if the initial resource is over-provisioned.

To develop a resource provisioning framework for the QoS-guaranteed resource provisioning problem, three perspectives need to be considered. The first is to formally describe the resource provisioning for cloud-based MapReduce in dynamic environments through a problem abstraction. The second is to design how to trigger different types of resource provisioning strategies derived from the problem abstraction. The third is to design algorithms to decide how much and how the resource is provisioned to cloud-based MapReduce computations.

Through addressing these three perspectives, this research differs from the existing efforts as it presents a novel event-driven framework for QoS-guaranteed resource provisioning. This major contribution of the development of the framework is incorporated with two other contributions: a new problem abstraction and three practical algorithms.

1. Novel event-driven framework: an event-driven triggering mechanism is developed for the resource provisioning framework. It detects the changes in the environment in the form of events. Through handling the events, the framework avoids the QoS violation due to resource under-provisioning caused by performance degradation in environments, and also potentially reduces the cost of using VMs due to resource over-provisioning caused by performance promotion in environments.

2. New problem abstraction: a dynamic optimization problem is abstracted from the QoS-guaranteed resource provisioning for cloud-based MapReduce in dynamic environments. The changes in environments have been considered in the dynamic optimization problem, so as to avoid QoS violation or resource waste caused by the changes.

3. Practical algorithms: resource provisioning algorithms are based on a solid theoretical analysis. They are designed to handle these events, which guarantees the QoS of cloud-based MapReduce while reducing the cost of using VMs.

6.2 Problem Characterization

Cloud-based MapReduce, a computation platform in cloud computing environments, utilizes various types of VMs from cloud computing to execute MapReduce computations. The MapReduce computations arrive on the platform at any time during the whole life of the platform, and their arrival times are not known in advance. To guarantee the QoS, cloud-based MapReduce

needs to meet the deadline of every arriving MapReduce computation. It also needs to minimize the total cost of using VMs, so as to minimize the operational cost of the platform.

Let $\mathbb{J}$ be the entire set of MapReduce computations arriving at cloud-based MapReduce. Let $J_k \in \mathbb{J}$ be a MapReduce computation, which is characterized as a tuple

$$< t_k^A, D_k, M_k^{CPU}, M_k^{Mem}, R_k^{CPU}, R_k^{Mem} > \tag{6.1}$$

In the tuple, t_k^A is the arrival time of J_k, and $0 \leq t_k^A \leq T_E$, where T_E is the end time of the platform; D_k is the deadline of J_k; M_k^{CPU} and M_k^{Mem} (R_k^{CPU} and R_k^{Mem}) respectively denote the CPU and memory requirements of the map (reduce) tasks of J_k.

Various types of VMs are used to execute the entire set of MapReduce computations, $\mathbb{J}$. Different types of VMs have different CPU capacities, memory capacities and prices. Let $\mathbb{V}$ be the entire set of the used VMs. Let $V_j \in \mathbb{V}$ be a used VM, which is characterized as a tuple

$$< V_j^{CPU}, V_j^{Mem}, p_j > \tag{6.2}$$

where V_j^{CPU} and V_j^{Mem} respectively represent the CPU and memory capacities of V_j, and p_j is the price (dollars per time unit) of V_j. V_j is charged by every time unit T_o, any partial utilization of V_j is charged as if the full time unit T_o was consumed. For example, if T_o is 3600 seconds, the cost of using V_j for 3601 seconds is equal to that for 7200 seconds.

To ensure every MapReduce computation in $\mathbb{J}$ is completed before its deadline, a set of workers are required to be provisioned to each MapReduce computation in $\mathbb{J}$. These Workers are categorized into two types: the map workers which execute map tasks, and the reduce workers which execute reduce tasks. Let $\mathbb{W}$ be the entire set of workers provisioned to the entire set of MapReduce computations $\mathbb{J}$. All the workers in $\mathbb{W}$ are assigned to the VMs in $\mathbb{V}$, and the VMs allocate CPU and memory resources to the workers. In addition, the allocated amount of the CPU and memory resources of a VM cannot exceed its CPU and memory capacity. Also, the CPU and memory resources allocated to a worker cannot be less than the CPU and memory requirements of the map/reduce tasks which the worker executes.

A worker $W_i \in \mathbb{W}$ is characterized as a tuple

$$< W_i^{CPU}, W_i^{Mem}, T_i^S, T_i^E, J_{W_i}, \alpha_i, V_{W_i} > \tag{6.3}$$

In this tuple, W_i^{CPU} and W_i^{Mem} respectively denote the amount of CPU and memory resources allocated to W_i. T_i^S is the time when W_i starts processing tasks. J_{W_i} denotes the index of the MapReduce computation to which W_i is provisioned, and $J_{W_i} = k$ means W_i is provisioned to execute J_k. α_i is a binary value indicating if W_i is a map worker or reduce worker, if $\alpha_i = 1$, W_i is a map worker. V_{W_i} indicates the index of the VM to which W_i is assigned, and $V_{W_i} = j$ means W is assigned to V_j.

Two functions are respectively introduced to quantify how worker provisioning affects the duration of a MapReduce computation and the usage time of a VM. One is $\mathrm{T}_k(\mathbb{W})$, which calculates the duration of J_k when $\mathbb{W}$ is given. Another is $\mathrm{T}_j(\mathbb{W})$, which quantifies the usage time of V_j when $\mathbb{W}$ is given. The values of these two functions change when the MapReduce computation environment changes.

6.3 Problem Formulation

The QoS-guaranteed resource provisioning problem is formulated as follows. Given a set of MapReduce computations, $\mathbb{J}$, the objective of the problem is to determine a set of workers, $\mathbb{W}$, and a set of VMs, $\mathbb{V}$, to minimize the total cost of using $\mathbb{V}$, that is

$$C = \sum_{j=1}^{|\mathbb{V}|} p_j \cdot \left\lceil \frac{\mathrm{T}_j(\mathbb{W})}{T_o} \right\rceil \tag{6.4}$$

subject to

$$t_k^A + \mathrm{T}_k(\mathbb{W}) \leq D_k \tag{6.5}$$

$$\sum_{i=1}^{n} (W_i^{CPU} \cdot \beta_i^j(t)) \leq V_j^{CPU} \tag{6.6}$$

$$\sum_{i=1}^{n} (W_i^{Mem} \cdot \beta_i^j(t)) \leq V_j^{Mem} \tag{6.7}$$

if $J_{W_i} = k$ and $\alpha_i = 1$, which means W_i is a map worker executing the map tasks of J_k,

$$W_i^{CPU} \geq M_k^{CPU} \tag{6.8}$$

$$W_i^{Mem} \geq M_k^{Mem} \tag{6.9}$$

if $J_{W_i} = k$ and $\alpha_i = 0$, which means W_i is a reduce worker executing the reduce tasks of J_k,

$$W_i^{CPU} \geq R_k^{CPU} \tag{6.10}$$

$$W_i^{Mem} \geq R_k^{Mem} \tag{6.11}$$

Detailed explanations about these constraints are given as follows. The constraint (6.5) ensures any MapReduce computation $J_k \in \mathbb{J}$ meet its deadline D_k, where $t_k^A + \mathrm{T}_k(\mathbb{W})$ indicates the completion time of J_k, and $1 \leq k \leq |\mathbb{J}|$; the constraints (6.6) and (6.7) make sure the allocated amount of CPU and memory resources of a VM do not exceed the CPU and memory capacities of that VM, respectively, where $\beta_i^j(t)$ indicates if W_i is located at V_j at the time t ($0 \leq t \leq T_E$); the constraints (6.8), (6.9) (or (6.10), (6.11)) guarantee every worker used to execute the map (or reduce) tasks of J_k has enough CPU and memory resources to execute those map (or reduce) tasks, where $1 \leq i \leq |\mathbb{W}|$.

From the computational point of view, the problem is a dynamic optimization one, as the objective function (6.4) and the deadline constraint (6.5) vary when the environment changes. When the environment experiences performance degradation, the values of the function $\mathrm{T}_k(\mathbb{W})$ in Equation (6.4) and the function $\mathrm{T}_j(\mathbb{W})$ in the constraint (6.5) increase; when the environment experiences performance promotion, the values of these two functions decrease.

6.4 Event-driven Resource Provisioning Framework

To solve the QoS-guaranteed resource provisioning problem for cloud-based MapReduce, we propose an event-driven resource provisioning framework in this section. Our framework detects all events that potentially cause any MapReduce computation hard deadline missing and unnecessary resource (VM) waste, and promptly handles those events. In this way, our framework can guarantee that the deadlines of those MapReduce computations running in our framework are met while minimizing the running cost of our framework.

6.4.1 Framework Architecture

As shown in Figure 6.1, the architecture of the event-driven resource provisioning framework has the following components:

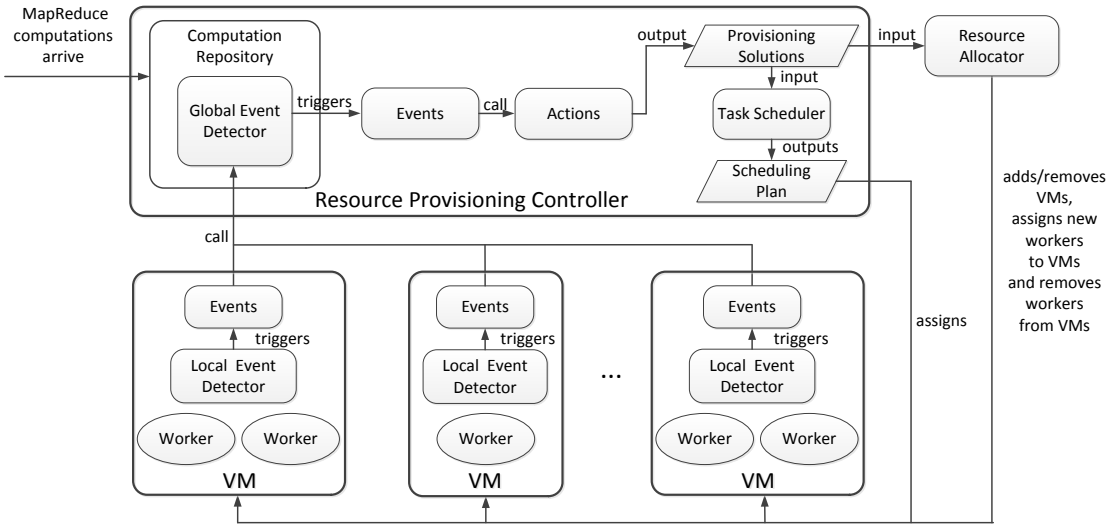


Figure 6.1: The architecture of event-driven resource provisioning framework

- Resource Provisioning Controller (RPC): determines how many and when workers and VMs need to be added to, or remove from, each of the admitted MapReduce computations.

- Computation Repository (CR): receives new MapReduce computations, and maintains the profiles and the computation progress of the admitted MapReduce computations.

- Local Event Detector (LED): detects events occurring on a VM. It knows the completion times of the map/reduce tasks executed on a VM.

- Global Event Detector (GED): detects events about admitted MapReduce computations. It has the global information about each of the admitted MapReduce computation, including the numbers of map and reduce tasks, computation progress, average task durations, maximal task durations, arrival time and deadline.

- Task Scheduler (TS): generates a scheduling plan, which specifies the assignment of the map/reduce tasks of a MapReduce computation to workers and the latest completion time of the map/reduce tasks before the MapReduce computation starts. This scheduling plan is assigned to each VM used by our framework before the MapReduce computation starts, and is updated when resource scaling actions are taken.

- Resource Allocator (RA): requests new VMs and assigns new workers to existing and new VMs, removes over-provisioned workers from VMs, and returns VMs in which no worker is assigned.

6.4.2 Event-driven Mechanism

The core of our framework is an event-driven mechanism. The actions of resource provisioning are driven by events. In our framework, there are eight different events and three resource provisioning actions as shown in Figure 6.2. This section explains what the eight events are how each of the events is generated and handled in our framework.

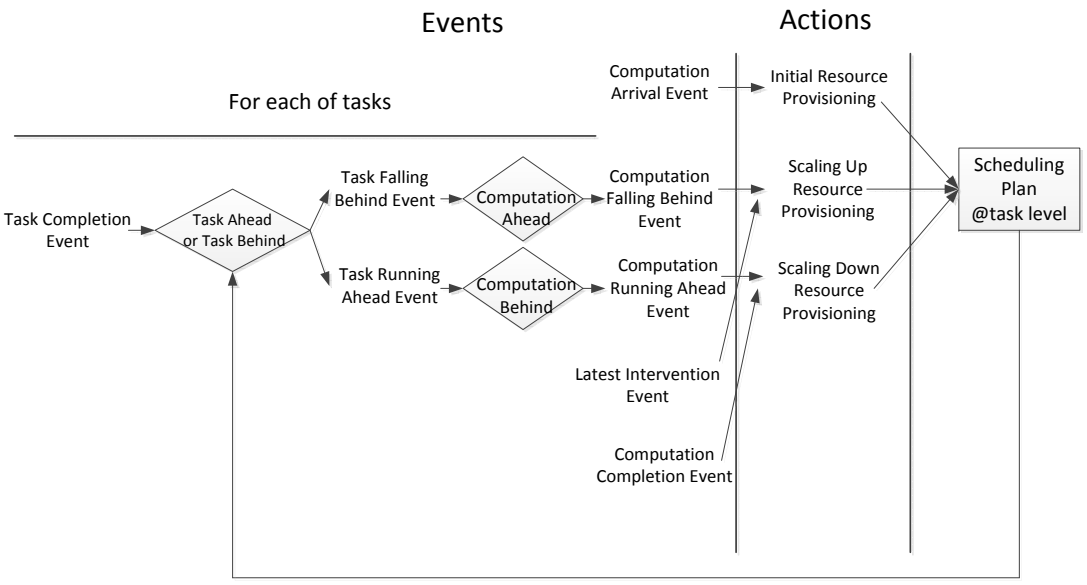


Figure 6.2: Events and actions

Events

This subsection formulates conditions for the generation of each of the events. The notations used in the event generation are listed as follows:

- J_k: an admitted MapReduce computation;
- t: the time of checking the trigger condition of an event;
- D: the deadline of a MapReduce computation;
- $\mathbb{W}_k$, $\mathbb{W}_k^+$, $\mathbb{W}_k^-$: the sets of pre-provisioned, new, and removed workers for J_k, respectively;
- M_0, R_0: the numbers of map and reduce workers in $\mathbb{W}_k$, respectively, $M_0 + R_0 = |\mathbb{W}_k|$;
- M_+, R_+: the numbers of map and reduce workers in $\mathbb{W}_k^+$, respectively, $M_+ + R_+ = |\mathbb{W}_k^+|$;

- M_-, R_-: the numbers of map and reduce workers in $\mathbb{W}_k^+$, respectively, $M_- + R_- = |\mathbb{W}_k^-|$;
- $\mathbb{V}$, $\mathbb{V}^+$, $\mathbb{V}^-$: the sets of existing, new, and removed VMs, respectively;
- m, r: the numbers of the map and reduce tasks of J_k, respectively;
- m_t, r_t: the uncompleted numbers of the map and reduce tasks of J_k at t, respectively;
- T_m^{avg}, T_m^{max}, T_r^{avg}, T_r^{max}:the average and maximal durations (measured in seconds) of the map and reduce tasks of J_k, respectively, and they are computed from past run or estimated by a profiling technology Verma et al. [2011a] in terms of task input sizes.
- T_o: the time delay caused by handling events;
- m_o^{min}, r_o^{min}: the lower bounds of the numbers of the map and reduce tasks completed within T_o, respectively;

Task Completion: A task completion event is generated by a LED of our framework whenever a map/reduce task finishes.

Task Falling Behind and Task Running Ahead: Whenever a task completion event occurs, the LED of our framework compares its actual completion time with its latest completion time which is decided by the scheduling plan. If its actual completion time is earlier than its planned latest completion time, then a task running ahead event is generated; if its actual completion time is later than its planned latest completion time, then a task falling behind event is generated.

Computation Arrival: A computation arrival event is generated once a MapReduce computation is received by the CR of our framework.

Computation Falling Behind: The generation condition of the computation falling behind event is checked by the GED of our framework once a task falling behind event is generated. The generation condition of a computation falling behind event is formulated in (6.12) if the MapReduce computation, J_k, is under the map phase, or in (6.13) if J_k completes its map phase.

$$a/M_0 + b/R_0 > c \tag{6.12}$$

$$1/R_0 > d/b \tag{6.13}$$

In the above two conditions, a, b, d and, d are respectively expressed by

$$a = (m_t - m_o^{min} - 1)T_m^{avg} \tag{6.14}$$

$$b = (r_t - r_o^{min} - 1)T_r^{avg} \tag{6.15}$$

$$c = D - t - T_o - T_m^{max} - T_r^{max} \tag{6.16}$$

$$d = D - t - T_o - \alpha T_m^{max} - T_r^{max} \tag{6.17}$$

where α is a binary number, and $\alpha = 0$ if $m_t \leq m_o^{min}$.

The condition (6.12) or (6.13) indicates that the upper bound of the completion time of the MapReduce computation, J_k, is later than its deadline under the current resource provisioning. Satisfying either of the two conditions means J_k is falling behind and cannot complete before the deadline under the current resource provisioning. More details and explanations about these conditions are seen in Resource Scaling-Up Condition Corollary presented by Xu et al. [2016].

Computation Running Ahead: The generation condition of a computation running ahead event is checked by the GED of our framework once a task running ahead event is generated. The generation condition of a computation running ahead event is formulated in (6.18) and (6.19) if the MapReduce computation, J_k, is under the map phase, or in (6.20) J_k completes its map phase.

$$\frac{a^{'}}{M_0} + \frac{b^{'}}{R_0 - 1} \leq c,\ \frac{a^{'}}{b^{'}} \geq \frac{M_0(M_0 - 1)}{R_0(R_0 - 1)} \tag{6.18}$$

$$\frac{a^{'}}{M_0 - 1} + \frac{b^{'}}{R_0} \leq c,\ \frac{a^{'}}{b^{'}} < \frac{M_0(M_0 - 1)}{R_0(R_0 - 1)} \tag{6.19}$$

$$\frac{d^{'}}{b^{'}} \geq \frac{1}{R_0} \tag{6.20}$$

where $a^{'}$, $b^{'}$, $c^{'}$ and $d^{'}$ are respectively expressed by the following equations:

$$a^{'} = (m_t - 1)T_m^{avg} \tag{6.21}$$

$$b^{'} = (r_t - 1)T_r^{avg} \tag{6.22}$$

$$c^{'} = D - t - T_m^{max} + T_m^{avg} - T_r^{max} + \gamma T_r^{avg} \tag{6.23}$$

$$d^{'} = D - t - T_r^{max} + \gamma T_r^{avg} \tag{6.24}$$

where γ is a binary number, and $\gamma = 0$ if $m_t > 0$.

The condition formulated in (6.18) and (6.19) or the condition formulated in (6.13) indicates that the upper bound of the completion time of the MapReduce computation, J_k, is earlier than its deadline under the current resource provisioning. Satisfying either of the two conditions means J_k is running ahead and one more workers are allowed to be removed. More details and explanations about these conditions are seen in Resource Scaling-Down Condition Corollary presented by Xu et al. [2016].

Latest Intervention: The GED of our framework utilizes a system timer to check if the latest intervention time formulated in Equation (6.25) comes. Once the latest intervention time achieves, the GED checks the generation condition of the latest intervention event, which is formulated in (6.26) where $m_t - m_o^{min}$ and $r_t - r_o^{min}$ respectively denote the maximal numbers of the remaining map and reduce tasks at the time $t + T_o$.

$$t = D - T_o - T_m^{max} - T_r^{max} \tag{6.25}$$

$$m_t - m_o^{min} > M_0 \text{ or } r_t - r_o^{min} > R_0 \tag{6.26}$$

The condition (6.26) indicates that the remaining map/reduce tasks at the latest intervention time are more than the map/reduce workers. That means the remaining map/reduce tasks cannot be completed in one wave. If the computation environment experiences the worst performance after the latest intervention time, the MapReduce computation, J_k, will fall behind schedule, and the upper bound of the completion time of J_k will be later than its deadline. According to the Latest Intervention Time Theorem Xu et al. [2016], when J_k falls behind schedule, the deadline could be missed no matter how many workers are, unless enough workers are provisioned before

the latest intervention time. To avoid missing the deadline due to intervening too late, enough resource should be provisioned when the condition (6.26) is satisfied.

Computation Completion: A computation completion event is generated when both the number of Map tasks, m_t, and the number of Reduce tasks, r_t become a zero.

6.4.3 Event Handling

An action must be immediately taken when each of the following events occurs as illustrated in Figure 6.2. When a Computation Arrival event occurs, the Initial Resource Provisioning action is taken; when a Computation Falling Behind event or a Latest Intervention event occurs, the Scaling Up Resource Provisioning action is taken; and when a Computation Running Ahead event or a Computation Completion event occurs, the Scaling Down Resource Provisioning action is taken. When any of the above-mentioned action is taken, a new scheduling plan for each of the involved MapReduce computations is generated by the TS of our framework, and is then assigned to each of the VMs for the MapReduce computations. The algorithms for the three resource provisioning actions will be discussed in detail in Section 6.5.

6.4.4 Advantages of Our Framework

Our event-driven framework guarantees the deadline of all admitted MapReduce computations while minimizing its running cost in a dynamical computation environment by promptly detecting and handling all the events that potentially lead to the violation of the deadlines of those MapReduce computations and the waste of resources (VMs). In addition, our event-driven framework does not suffer from the overheads incurred by the periodical checking of the status of each of the admitted MapReduce computations.

6.5 Algorithms

Three algorithms have been developed to handle the computation arrival event, computation falling behind event, computation running ahead event, latest intervention event and computation completion event.

6.5.1 Scaling Up Algorithm

Scaling Up Algorithm (SUA) is applied to handle the computation falling behind and latest intervention events. SUA determines the minimum number of workers that needs to add to the MapReduce computation in order to catch its deadline. If the spare capacities of existing VMs are not sufficient to host the new workers, then new VMs are used. If this is the case, the cost of hiring new VMs is minimized. Algorithm 6 is the description of SUA.

The inputs of SUA are the set of the pre-provisioned workers for the MapReduce computation J_k that needs to scale up its resources, $\mathbb{W}_k$, and the set of existing VMs, $\mathbb{V}$. The output is a set of new workers provisioned to J_k, $\mathbb{W}_k^+$.

Algorithm 6 SUA

1: determine the numbers of new map and reduce workers, M_+ and R_+, respectively;
2: add M_+ map workers and R_+ reduce workers to $\mathbb{W}_k^+$;
3: output $\mathbb{W}_k^+$.

Firstly, SUA determines the minimal number of new map workers and the minimal number of new reduce workers to be added into $\mathbb{W}_k^+$, M_+ and R_+, which minimizes the resource consumption while guaranteeing the deadline of J_k. The determination of M_+ and R_+ depends on the type of the triggered event. It the event is a computation falling behind event, M_+ and R_+ are obtained by addressing a constrained optimization problem. In particular, if J_k is under map phase, the problem is formulated as a Non-Linear Integer Programming (NLIP):

$$\begin{aligned} \min Z &= M_+ \cdot (a^{CPU} \cdot M_k^{CPU} + a^{Mem} \cdot R_k^{mem}) \\ &+ R_+ \cdot (a^{CPU} \cdot M_k^{CPU} + a^{Mem} \cdot R_k^{mem}) \end{aligned} \tag{6.27}$$

subject to

$$\frac{a}{M_0 + M_+} + \frac{b}{R_0 + R_+} \leq c \tag{6.28}$$

$$M_+ \geq 0, R_+ \geq 0 \tag{6.29}$$

$$M_+, R_+ \in \mathbb{Z} \tag{6.30}$$

where a^{CPU} and a^{Mem} respectively represent the surrogate weights for CPU and memory resources, and $a^{CPU} + a^{Mem} = 1$; a, b, and c are respectively expressed by Equations (6.14), (6.15) and (6.16).

In this formulation, the objective function (6.27) calculates the minimum amount of resource consumption by the workers to be added. The constraint (6.28) ensures J_k is completed before the deadline, which is derived by the QoS-guaranteed Scaling-up theorem, presented in the paper of Xu et al. [2016]; the constraints (6.29) and (6.30) make sure M_+ and R_+ are non-negative integers. This problem can be easily solved by many mathematical tools; one example is Lingo [2015]. The detailed procedure for the NLIP problem is presented in the paper of Garfinkel and Nemhauser [1972].

If J_k completes the map phase, M_+ and R_+ are also derived by addressing the constrained optimization problem. However, the constraint (6.28) of the problem is replaced by (6.31), which is derived by the QoS-guaranteed Scaling-up theorem, presented in the paper of Xu et al. [2016].

$$\frac{1}{R_0 + R_+} \leq \frac{d}{b} \tag{6.31}$$

Then the solution to the problem can be immediately derived, that is $M_+ = 0$ and $R_+ = \lceil b/d - R_0 \rceil$, where b and d are expressed by Equations (6.15) and (6.17), respectively.

In addition, if the event is a latest intervention event, M_+ and R_+ are derived by addressing the constrained optimization problem. But the constraint (6.28) of the problem is replaced by (6.32).

$$M_0 + M_+ \geq m_t - m_o^{min},\ R_0 + R_+ \geq r_t - r_o^{min} \tag{6.32}$$

The constraint (6.32) is derived from the analysis of the generation conditions (6.25) and (6.26) of the Latest Intervention event, and this constraint ensures the deadline of J_k is guaranteed even when the environment experiences the worst performance. Then the solution to the problem can be immediately derived, that is $M_+ = \lceil m_t - m_o^{min} - M_0 \rceil$ and $R_+ = \lceil r_t - r_o^{min} - R_0 \rceil$.

Secondly, SUA adds the M_+ map workers and R_+ reduce workers to $\mathbb{W}_k^+$. Finally, SUAoutputs $\mathbb{W}_k^+$.

6.5.2 Scaling Down Algorithm

Scaling Down Algorithm (SDA) is applied to handle the computation running ahead and computation completion events. SDA removes a set of workers from a MapReduce computation J_k which is running ahead or has completed, and therefore reduces the number of VMs to minimize the cost of hiring VMs while still meeting the deadline of J_k. Algorithm 7 is the descriptions of SDA.

The inputs of SDA are the set of pre-provisioned workers for J_k, $\mathbb{W}_k$, and the set of existing VMs, $\mathbb{V}$. The outputs are a set of workers to be removed, $\mathbb{W}_k^-$, and a set of VMs to be removed, $\mathbb{V}^-$.

Algorithm 7 SDA

1: determine the maximal number of map workers, M_-, and the maximal number of reduce workers, R_-, which can be removed without affecting the deadline of the MapReduce computation;
2: sort those VMs on which the workers of the MapReduce computation are placed by the descending order of the ratio of the resources occupied by the workers of the MapReduce computation and the resources occupied by the workers of the other MapReduce computations;
3: from the first VM to the last VM, remove M_- map workers from those VMs and put them into $\mathbb{W}_k^-$;
4: from the first VM to the last VM, remove R_- reduce workers from those VMs and put them into $\mathbb{W}_k^-$;
5: from the first VM to the last VM, put all the VMs that do not have any workers into $\mathbb{V}^-$;
6: output $\mathbb{W}_k^-$ and $\mathbb{V}^-$;

Step 1 of SDA is to determine the maximal numbers of map and reduce workers to be removed, M_- and R_-, respectively, while guaranteeing the deadline of J_k. M_- and R_- are derived by addressing a constrained optimization problem, which is formulated as a Non-Linear Integer Programming (NLIP) problem as follows:

$$\begin{aligned} \max Z = & M_- \cdot (a^{CPU} \cdot M_k^{CPU} + a^{Mem} \cdot R_k^{mem}) \\ & + R_- \cdot (a^{CPU} \cdot M_k^{CPU} + a^{Mem} \cdot R_k^{mem}) \end{aligned} \tag{6.33}$$

subject to

$$\frac{a^{'}}{M_0 - M_-} + \frac{b^{'}}{R_0 - R_-} \leq c^{'} \tag{6.34}$$

$$0 \leq M_- < M_0, 0 \leq R_- < R_0 \tag{6.35}$$

$$M_-, R_- \in \mathbb{Z} \tag{6.36}$$

d where a^{CPU} and a^{Mem} respectively represent the surrogate weights for CPU and memory resources, and $a^{CPU} + a^{Mem} = 1$; $a^{'}$, $b^{'}$, and $c^{'}$ are respectively expressed by Equations (6.21), (6.22) and (6.23).

In this formulation, the objective function (6.33) calculates the maximal number of VMs to be removed. The constraint (6.34) ensures J_k is completed before its deadline, which is determined by the QoS-guaranteed Scaling-down theorem presented in Xu et al. [2016]; the constraints (6.35) and (6.36) make sure M_- and R_- are non-negative integers, and at least one map worker and one reduce worker are left. The detailed procedure for the NLIP problem is presented in Garfinkel and Nemhauser [1972].

If J_k completes the map phase, constraints (6.34) and (6.35) are replaced by (6.37) and (6.38), which are derived from the QoS-guaranteed Scaling-down theorem presented in Xu et al. [2016].

$$\frac{1}{R_0 - R_-} \leq \frac{d^{'}}{b^{'}} \tag{6.37}$$

$$0 \leq M_- \leq M_0,\ 0 \leq R_- < R_0 \tag{6.38}$$

Then the solution to the problem can be immediately derived, that is $M_- = M_0$ and $R_- = \lfloor R_0 - b^{'}/d^{'} \rfloor$, where b and d are expressed by Equations (6.15) and (6.17), respectively.

If the event is a computation completion event, just let $M_- = M_0$ and $R_- = R_0$, which means that all the map and reduce workers for J_k are moved, such that the number of VMs used by the remaining workers is minimized.

Step 2 is to sort those VMs on which the workers of the MapReduce computation are placed by the descending order of the ratio of the resources occupied by the workers of the MapReduce computation and the resources occupied by the workers of the other MapReduce computations.

Steps 3 and 4 remove the M_- map workers and R_- reduce workers the from those VMs, and put them into $\mathbb{W}_k^-$. Step 5 puts all the VMs that do not have any workers into $\mathbb{V}^-$. Finally, Step 6 outputs the set of workers to be removed, $\mathbb{W}_k^-$, and the set of VMs to be removed, $\mathbb{V}^-$.

6.5.3 Initial Provisioning Algorithm

Initial Provisioning Algorithm (IPA) is applied to handle the new computation arrival event. When a new MapReduce computation arrives, IPA generates a set of workers for the new MapReduce computation such that the MapReduce computation can meet its deadline while consuming minimum resource based on the current performance of our framework, and then uses a MapReduce placement algorithm to find a set of new VMs of various types and assign the new workers to the new and existing VMs while minimizing the cost of hiring the VMs. The details of SUA are described in Algorithm 8.

The input of IPA is a set of the existing VMs, $\mathbb{V}$. The outputs are the sets of new workers initially provisioned to any MapReduce computation $J_k \in \mathbb{J}_{new}$, $\mathbb{W}_1^+, \mathbb{W}_2^+, \cdots, \mathbb{W}_{|\mathbb{J}_{new}|}^+$, and a set of new VMs used to load the new workers, $\mathbb{V}^+$.

IPA iterates $|\mathbb{J}_{new}|$ times (Steps 1-5) to generates a set of workers, $\mathbb{W}^+$, initially provisioned to every MapReduce computation in $\mathbb{J}_{new}$. In the k^{th} iteration ($1 \leq k \leq |\mathbb{J}_{new}|$), IPA implements a similar procedure as Steps 1 and 2 of SUA to generate a set of workers initially provisioned to the MapReduce computation $J_k \in \mathbb{J}_{new}$ and then adds them into $\mathbb{W}^+$.

Finally, Step 6 of IPA outputs the sets of new workers initially provisioned to any MapReduce computation $J_k \in \mathbb{J}_{new}$, $\mathbb{W}_1^+, \mathbb{W}_2^+, \cdots, \mathbb{W}_{|\mathbb{J}_{new}|}^+$, and the set of new VMs used to load the new workers, $\mathbb{V}^+$.

Algorithm 8 IPA

1: **for** $k = 1$ to $|\mathbb{J}_{new}|$ **do**
2: determine the CPU and memory requirements of the new map workers and new reduce workers to be provisioned to new MapReduce computation $J_k \in \mathbb{J}_{new}$;
3: determine the minimal number of map workers and the minimal number of reduce workers, M_+^k and R_+^k, respectively such that J_k can finish its computation before its deadline;
4: add M_+^k map workers and R_+^k reduce workers to $\mathbb{W}^+$;
5: **end for**
6: output $\mathbb{W}_1^+, \mathbb{W}_2^+, \cdots, \mathbb{W}_{|\mathbb{J}_{new}|}^+$, and $\mathbb{V}^+$.

6.6 Validation and Evaluation

The QoS-guarantee of our event-driven resource provisioning framework was validated and the performance of our event-driven resource provisioning framework was evaluated by simulation. The simulation setup is explained in detail first before the simulation results are shown and discussed in this section.

6.6.1 Simulation Setup

The validation and evaluation were conducted by simulation. The simulation mimicked the execution of a MapReduce computation in a cloud environment where the computation performance varied over time. The details about the simulation are as follows.

Simulation Inputs

The inputs of the simulation included the information about MapReduce computations, VMs, and environment variations. However, our framework did not use any information about environment variations in the simulation.

The MapReduce computations were used in the simulation included *WordCount*, *Sort*, *NutchIndexing* and *K-means*, which were selected from a popular MapReduce benchmark suit namely HiBench Huang et al. [2010]. During the simulation period, there were 315 instances of the four MapReduce computations submitted to our framework in the first 3600 seconds. The 315 instances of the MapReduce computation instances were randomly generated and their submission times followed a Poisson distribution with the mean of 0.0628 in the light of the research results presented in Zaharia et al. [2009]. In the simulation, our framework did not know the number of instances of MapReduce computation that would be submitted or their submission times in advance.

There were three different configurations for each instance of the four types of MapReduce computations: *small*, *medium*, and *large*. Details about each of the three configurations are shown in Table 6.1. The information was based an observation of Facebook over a week in October 2009 Zaharia et al. [2009]. The deadlines for the small, medium, and large MapReduce computations were set to 400 seconds, 600 seconds, and 800 seconds, respectively, in the

Table 6.1: Configurations of MapReduce computations

Type	# of Map Tasks	# of Reduce Tasks	Proportion
Small	15	4	68 %
Medium	105	40	21 %
Large	405	100	11 %

Table 6.2: The VM types from Amazon EC2

VM Type	CPUs (#)	Memory (GB)	Price ($/hour)
m3.medium	1	3.15	0.098
m3.large	2	6.9	0.196
m3.xlarge	4	14.4	0.392
m3.2xlarge	8	29.4	0.784

simulation.

The resource requirements and execution times of map/reduce tasks were obtained as follows. Each instance of the four types of MapReduce computations ran on a local cluster of VMs (six 2.40GHz cores and 8 GB Memory) with different configurations presented by Table 6.2, respectively. The resource requirements of the map/reduce tasks for each MapReduce computation instance were then obtained through profiling. Meanwhile, the execution times of the map/reduce tasks of each instance were also observed, which were assumed to be unknown by our framework in advance.

Next, the information about VMs was presented in Table 6.2, which were based on Amazon EC2 offerings. In addition, the boost time of each VM was configured as 97 seconds Mao and Humphrey [2011], according to Mao and Humphrey's observation from Amazon EC2 Cloud Amazon [2015],

Finally, the information about environment variation was described as follows. The environment variations were defined in the form of a sequence of [time, variation ratio] pairs. The variation ratio was defined as the ratio of the longest execution time of a map/reduce task and the initially observed execution time of that map/reduce task. Using this variation sequence, the same task could have different execution time at different time points, which mocked the influence of environment variations on MapReduce computations. For example, if the initially observed execution time of a task was 100 seconds and a pair was $< 300, 1.2 >$, the execution time of that task was 100 seconds before the time point of 300 seconds, but it was 120 seconds after the time point of 300 seconds. In this simulation, the number of variations (or the length of the sequence) was uniformly distributed between 50 and 100, the time point of variation

was uniformly distributed between 0 and 18000 seconds, and the variation ratio was uniformly distributed between 1 to 1.3 according to the observation from Amazon EC2 Farley et al. [2012].

Simulation Process

Given the simulation inputs, the simulation procedure commenced. Once a MapReduce computation arrived, the resource provisioning framework determined the required numbers and types of new VMs. The simulation mocked that the request of new VMs consumed 97 seconds. The framework then determined the number of workers and the placement of the workers on VMs. After that, the framework assigned the map/reduce tasks of the MapReduce computation to the workers. Once a worker finished a task, the framework assigned the next one to that worker until all the tasks completed. The execution time of each map/reduce task was initially the observed execution time, but it varied based on the environment variation information. When all of the map/reduce tasks of the MapReduce computation completed, the workers for this MapReduce computation were removed and then the idle VMs were also removed.

Specially, if the framework was a dynamical one, the simulation mocked how to scaling up/down resource at runtime. If resource scaling up was required, the framework firstly determined the required numbers and types of new VMs, and decided the numbers of new workers and their placement on VMs. After that, the framework assigned the remaining map/reduce tasks to the new workers. But, the workers on the new VMs did not start running tasks until 97 seconds passed. If resource scaling down was required, the framework removed the redundant workers, the running tasks on these workers were back in complete and waited for re-assignment.

Simulation Output

When the simulation finished, it displayed the actual execution time of every MapReduce computation instance and the deadline for each of the MapReduce computation instance. From this set of output, we know if there was an instance of MapReduce computation did not finish before its deadline. In addition, it also showed the usages for each type of the VMs and the total cost for hiring the VMs.

6.6.2 Experimental Results

The results of the validation and evaluation for the event-driven resource provisioning framework are presented in the following.

Validation of QoS-guarantee

The QoS-guarantee of the framework was validated by simulating the execution of a large number of MapReduce computations in our framework under various circumstances. If all the MapReduce computations met their deadline, then we claim the framework is QoS-guaranteed; otherwise, the framework was not QoS-guaranteed.

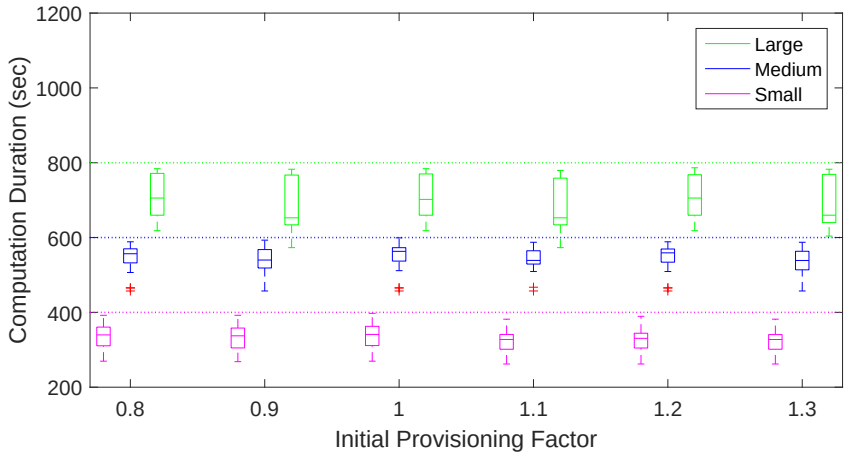


Figure 6.3: Distributions of the computation times of small, medium and large computations when deadline tightness is 1.0

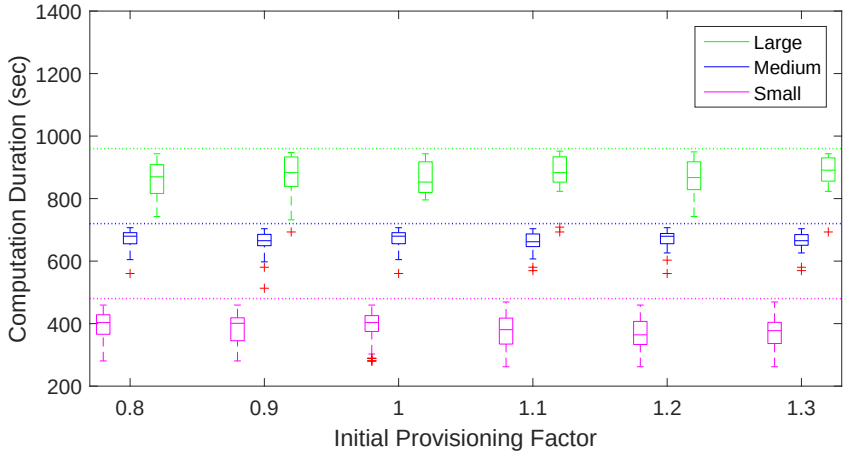


Figure 6.4: Distributions of the computation times of small, medium and large computations when deadline tightness is 1.2

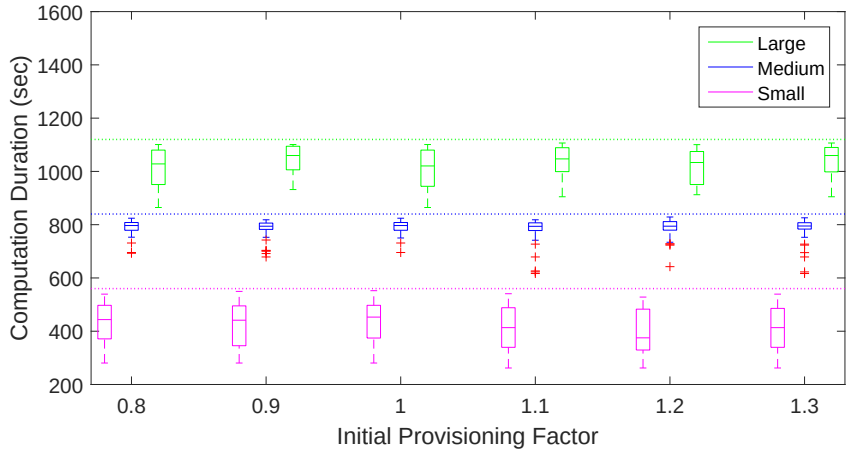


Figure 6.5: Distributions of the computation times of small, medium and large computations when deadline tightness is 1.4

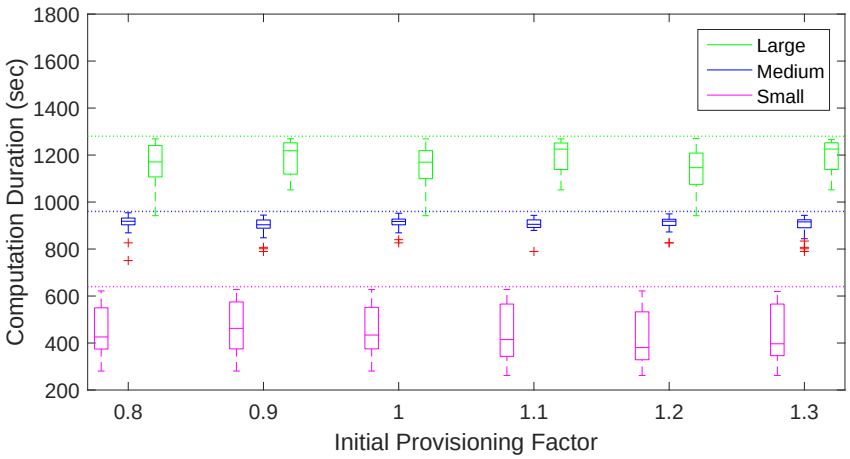


Figure 6.6: Distributions of the computation times of small, medium and large computations under different deadline tightness when deadline tightness is 1.6

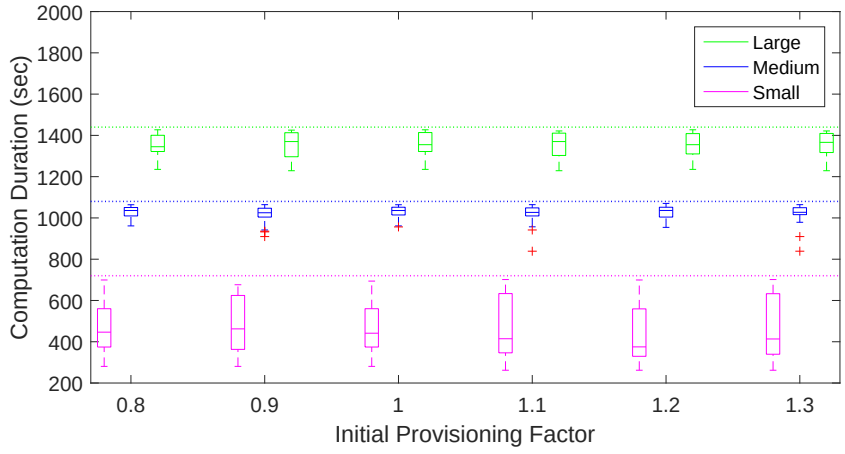


Figure 6.7: Distributions of the computation times of small, medium and large computations under different deadline tightness when deadline tightness is 1.8

Figure 6.3−Figure 6.7 describe the duration distributions of the small, medium, large MapReduce computations. In the figures, the box represents the ranges of the variations of the MapReduce durations, the line under the box is the shortest MapReduce duration, and the line or cross above the box is the longest MapReduce duration. The *deadline tightness* is the ratio of a deadline to its baseline. For example, for a small computation whose deadline baseline is 400 seconds, its deadline is 480 seconds if the tightness is 1.2. The *initial provisioning factor* indicates the ratio resource amount initially provisioned to its minimum amount to meet the deadline. For example, for a MapReduce computation initially requiring 10 workers to meet its deadline, it will be provisioned to eight workers if the initial provisioning factor is 0.8. Thus, the initial resource amount is under-provisioned if the factor is smaller than 1, while the initial resource amount is over-provisioned if the factor is larger than 1.

As shown in Figure 6.3−Figure 6.7, all three categories of computations always met their respective deadlines no matter how much the initial resource amount was provisioned or how tight the deadlines were. Therefore, the QoS-guarantee of the event-driven resource provisioning framework was validated.

Evaluation of Performance

The second experiment was to evaluate the performance of the event-driven resource provisioning framework on the cost of using VMs. The evaluation was conducted through comparing the event-driven resource provisioning framework with the static and periodic ones in five cases with different deadline tightness.

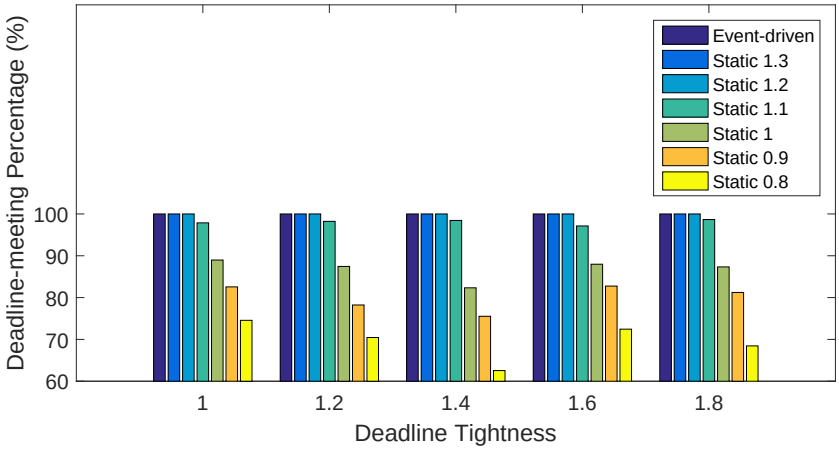


Figure 6.8: Performance of the static and event-driven resource provisioning frameworks on deadline-meeting percentages

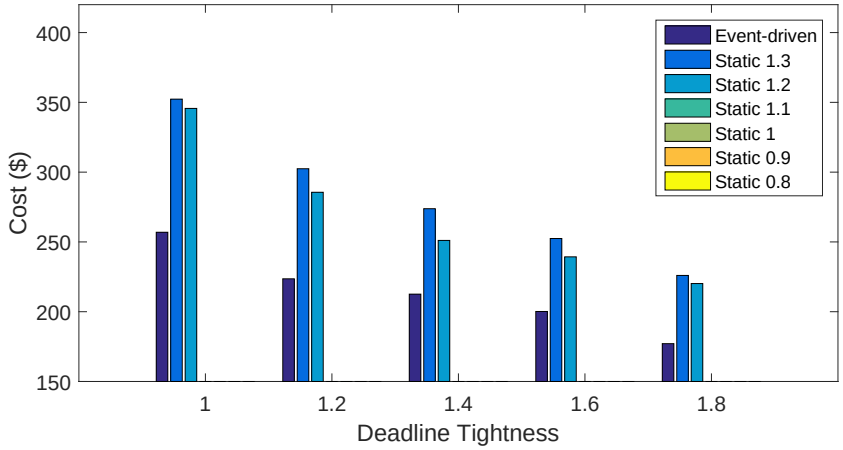


Figure 6.9: Performance of the static and event-driven resource provisioning frameworks on cost of using VMs

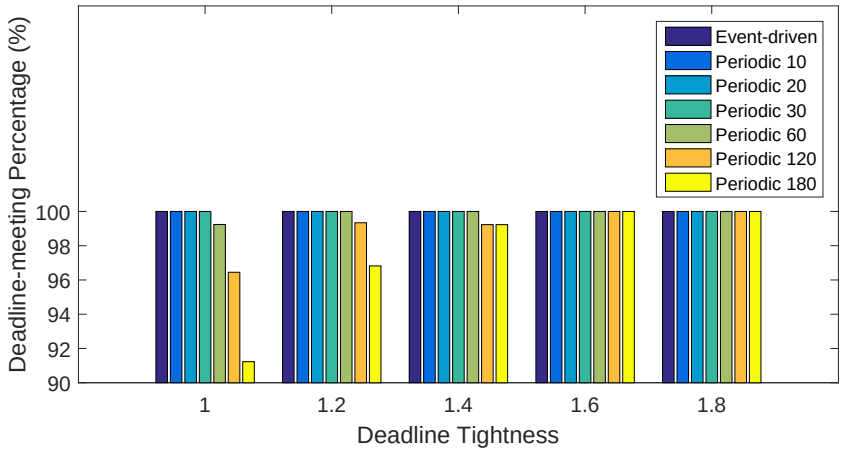


Figure 6.10: Performance of the static and event-driven resource provisioning frameworks on deadline-meeting percentages

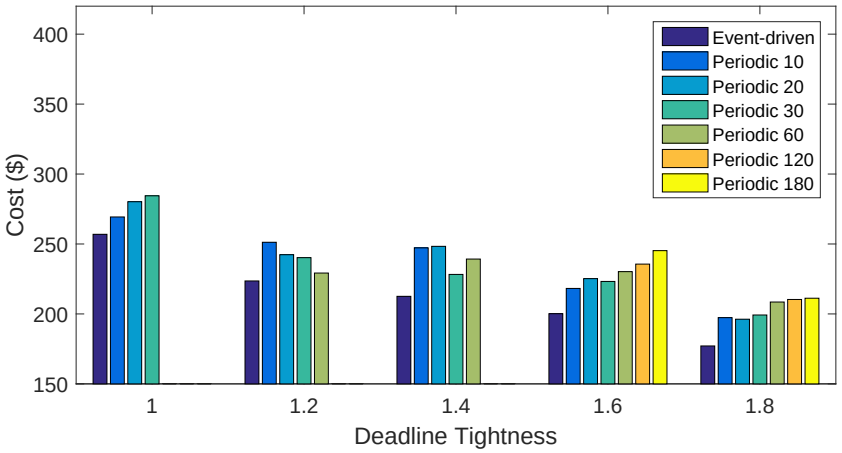


Figure 6.11: Performance of the static and event-driven resource provisioning frameworks on cost of using VMs

Figure 6.8−Figure 6.11 illustrate the comparison results of the performance of the event-driven and static resource provisioning frameworks, and the comparison results of the performance of the event-driven and periodic resource provisioning frameworks, respectively. In the figures, *Event-driven* indicates the event-driven resource provisioning framework. *Static* x denotes the static resource provisioning framework, and x is the initial provisioning factor. *Periodic* x denotes the periodic resource provisioning framework configuring the period as x. The periodic resource provisioning frameworks respectively apply IPA, SUA, and SDA to initialize, scale up, and scale down the resource provisioning for the MapReduce computations within each period. *deadline-meeting percentage* represents the percentage of the MapReduce computations meeting its deadlines in the workload. If a framework cannot ensure the deadline-meeting percentage is 100 percent, it means the framework cannot ensure every MapReduce computation meet its deadline and the cost of using VMs under that framework will not be presented in the figures.

As shown in Figure 6.8, using the static resource provisioning framework, the deadline-meeting percentage always was lowered as the initial resource amount decreased, no matter how the deadline tightness changed. On the contrary, using the event-driven framework, the deadline-meeting percentage always was 100 percent when the deadline tightness changed. Meanwhile, as shown in Figure 6.9 as the deadline tightness varied, the cost of using VMs under the event-driven resource provisioning framework was $15.3 - 27.1$ percent less than that under the static framework.

As shown in Figure 6.10, if the period of the periodic resource provisioning framework was too long, the deadlines of some MapReduce computations were missed as the intervention was too late. When the deadline tightness varied, the optimum period of the periodic framework also changed. The shorter period did not mean less cost of using VMs. As shown in Figure 6.11, as the deadline tightness varied, the cost of using VMs under the event-driven resource provisioning framework was $2.5 - 15.8$ percent less than that under the periodic framework.

The third experiment was to evaluate how the number of VMs used by the event-driven framework would change when the number of running MapReduce computations varied over time. In this experiment, we set the values in the variation sequence to be one, eliminating the influence of environment performance fluctuation on the number of VMs used by our framework. We used only one type of VMs in Table 6.2, xlarge, in this experiment.

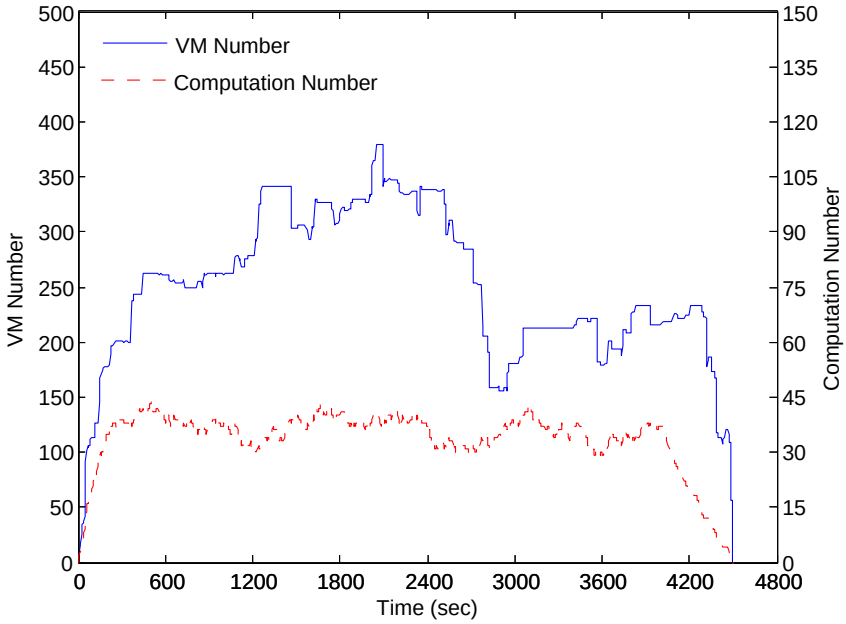


Figure 6.12: The number of using VMs when the number of running MapReduce computations changes over time

Figure 6.12 shows the experimental results. In this figure, the X-axis indicates the runtime of our framework; the left Y-axis indicates the number of VMs used at a time point and the right Y-axis indicates the number of running MapReduce computations at that time. As shown in the figure, sometimes the number of VMs used by our framework increased when the number of running MapReduce computations increased. But, sometimes the number of VMs used by our framework did not increase when the number of running MapReduce computations increased. For example, from 1800 seconds to 2400 seconds, the number of running MapReduce computations changed slightly but the number of VMs increased dramatically. The reason for this was the new MapReduce computations were larger and therefore required more resources than those just completed MapReduce computations during the period. Therefore, the number of VMs used by our framework is not proportional to the number of running MapReduce computations. The number of VMs used by our framework depends on the number of running MapReduce computations and the requirements of the running MapReduce computations. A smaller number of large MapReduce computations may use more VMs than a large number of small MapReduce computations. In addition, the number of VMs used by our framework also depends on the capacities of the VMs. For the same MapReduce computations, if more powerful VMs are used, then the total number of VMs used by our framework would be smaller. If less powerful VMs used by our framework, then the total number of VMs used by our framework would be larger.

The fourth experiment was to evaluate how the number of VMs used by our framework would change when the environment performance varied over time. In this experiment, we randomly generated 10 different types of medium MapReduce computations with the same deadlines of 1800 seconds. They were submitted to our framework at the same time and no other MapReduce computations were submitted to our framework, to avoid the influence of MapReduce computation submissions on the number of VMs used by our framework. We used only one type of VMs in Table 6.2, that is xlarge.

Figure 6.13 shows the experimental results. In this figure, the X-axis indicates the runtime of the 10 MapReduce computations; the left Y-axis indicates the number of VMs used by our framework at a time point and the right Y-axis indicates the variation ratio defined in the variation sequence; the increase/decrease in the variation ratio means there was a performance degradation/promotion in the environment. The 10 MapReduce computations commenced at the time point of 0 seconds. The MapReduce computations experienced two environment performance degradations, one at the time points of 60 seconds and another at 640 seconds, and our framework added three and five VMs, respectively, to response the computation environment's degradation. By contrast, they experienced environment performance promotion at the time points of 460 seconds and 960 seconds, but our framework did not shut down VMs, since the VMs might store the intermediate data for the MapReduce computations and could not be shut down until the MapReduce computations completed. In fact, our framework removed some workers and reserved resource space for resource scaling-up in future, which was illustrated by the third environment performance degradation occurring at the time point of 1150 seconds. The framework did not add VMs when this performance degradation occurred, as enough resource space was released from the previous resource scaling down and could be used for the newly added workers.

6.7 Summary of Chapter

This chapter has developed an event-driven resource provisioning framework for cloud-based MapReduce, and has validated the QoS-guarantee of the event-driven resource provisioning framework by experiments. The experimental results have shown that the event-driven framework always ensured every MapReduce computation met its deadline as the deadline tightness

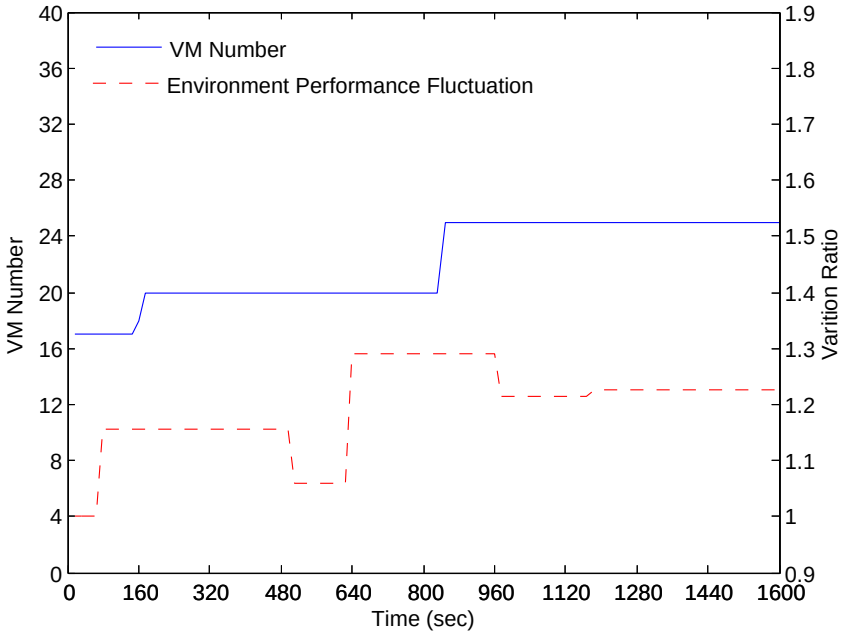


Figure 6.13: The number of using VMs when the environment performance changes over time

changed. This chapter has also used experiments to evaluate the performance of the event-driven frameworks on saving the cost of using VMs. The experimental results have shown that the event-driven resource provisioning framework saved $15.3 - 27.1$ percent more cost than the static resource provisioning frameworks, and $2.5 - 15.8$ percent more cost than the periodic resource provisioning frameworks.

Chapter 7

Conclusion and Future Work

This chapter summarizes the work of this thesis, then recaps the major contributions of this work, and finally discusses our future work.

7.1 Summary of Research

This thesis has studied the QoS-guaranteed resource provisioning problem in cloud-based MapReduce. Four research problems involved in the research include the theoretical study of QoS-guaranteed resource provisioning for cloud-based MapReduce, the cloud-based MapReduce placement (MRP) problem, cloud-based MapReduce consolidation (MRC) problem and the development of the QoS-guaranteed resource provisioning framework. The studies of the first three research problems lay the foundation for the study of the fourth research problem. Based on the outcomes from the first three research problems, the study of the fourth research problem finally solves the QoS-guaranteed resource provisioning problem in cloud-based MapReduce. The investigations of the four research problems and how they have answered the research questions have respectively been presented in four chapters as follows.

Chapter 3 has presented the theoretical study of QoS-guaranteed resource provisioning for cloud-based MapReduce. In that chapter, the impact of resource scaling on MapReduce computation time has first been quantified. Then the QoS-guaranteed resource provisioning problem has been characterized through a non-linear transformation. By that non-linear transformation, the theoretical study has been greatly simplified. After that, sufficient conditions on timing and amount of resource scaling-up/down to guarantee QoS, summarized as three theorems and two

corollaries, have been derived and also have been proven by strict mathematical deductions. Finally, case studies have been conducted to demonstrate the theoretical results.

Chapter 4 has investigated the MRP problem, and has proposed a new MapReduce placement algorithm to solve the MRP problem successfully. In that chapter, the MRP problem has been formulated as a constrained optimization problem. Then the new MapReduce placement algorithm has been designed in two phases. The first phase is to generate a set of promising placement patterns; the second phase is to find a combination of the generated placement patterns to build a solution to the MRP problem. The new algorithm answers the research question about how to determine the right size and type of VMs and the placement of MapReduce computations on VMs. After that, the performance of the new algorithm on the operational cost of cloud-based MapReduce has been evaluated by comparing it with a popular homogeneous placement algorithm and three heterogeneous placement algorithms. Meanwhile, the scalability of the new algorithm has been demonstrated by case studies.

Chapter 5 has studied the MRC problem and has designed a new MapReduce consolidation algorithm to solve the MRC problem successfully. In that chapter, the MRC problem has been formulated as a bio-objective optimization problem. Then the new MapReduce consolidation algorithm has been designed, which consists of three phases. The first phase of the algorithm is to find a subset of VMs required to be consolidated, the second phase is to generate a set of promising placement patterns, and the third phase of the algorithm is to find a combination of the generated placement patterns to build a solution to the MRC problem. The new algorithm answers the research question about how to consolidate MapReduce computations on VMs. Then the effectiveness of the new algorithm has been demonstrated by comparing the operational cost of cloud-based MapReduce using that algorithm and not using that algorithm. Also, the performance of the new algorithm on saving the operational cost of cloud-based MapReduce has been evaluated by comparing it with a baseline consolidation algorithm. In addition, the scalability of the new algorithm has been evaluated by experiments.

Chapter 6 has developed the QoS-guaranteed resource provisioning framework. The QoS-guaranteed resource provisioning problem has been formulated as a dynamic optimization problem and has been solved by a novel resource provisioning framework. In the framework, the trigger mechanism has identified seven events and their trigger conditions based on the theoretical timing conditions of QoS-guaranteed resource provisioning. That has answered the research

question about how to determine the timing of resource provisioning. Each of the events activates a type of resource provisioning algorithm. Based on the theoretical amount conditions of QoS-guaranteed resource provisioning, the resource provisioning algorithms have calculated the amount of QoS-guaranteed resource provisioning. That has answered the research question about how to determine the amount of resource provisioning. The framework has adopted a resource allocator. The resource allocator has applied the new MapReduce placement algorithm to answer the research question: how to choose the number and type of VMs and the placement of MapReduce computations on VMs? It has also adopted the new MapReduce consolidation algorithm to answer the research question: how to consolidate MapReduce computations on VMs?

Having solved the four research problems, all research questions have been answered. Consequently, the issue about QoS-guaranteed resource provisioning for cloud-based MapReduce has been addressed successfully.

7.2 Major Contributions

Having addressed the four research problems, this thesis makes the following four contributions.

7.2.1 New Theoretical Results of QoS-guaranteed Resource Provisioning for Cloud-based MapReduce

Through the theoretical analysis of QoS-guaranteed resource provisioning for cloud-based MapReduce, new theoretical results have been derived. This work is the first attempt to the theoretical study of QoS-guaranteed resource provisioning for cloud-based MapReduce and current research has rarely given such a detailed theoretical study.

These theoretical results, laying the theoretical foundation for QoS-guaranteed resource provisioning for cloud-based MapReduce, are used for designing the new QoS-guaranteed resource provisioning framework. The theoretical results guide the framework to determine the amount and timing of QoS-guaranteed resource provisioning. Only when the amount or timing of resource provisioning satisfies the sufficient conditions presented by the theoretical results can the QoS of cloud-based MapReduce be guaranteed.

7.2.2 New MapReduce Placement Algorithm for Cloud-based MapReduce

A new MapReduce placement algorithm has been designed, successfully solving the MRP problem. This new placement algorithm is totally different from these current algorithms for the cloud-based MapReduce consolidation problem, and has a better performance than the current algorithms on saving the operational cost of cloud-based MapReduce.

The new placement algorithm saves more operational cost of cloud-based MapReduce than most current algorithms. Most of the current algorithms are categorized as homogeneous ones which assign MapReduce computations on homogeneous VMs or assign the same number of map/reduce workers on each VM. However, the new placement algorithm is a heterogeneous one, which allows using heterogeneous VMs and heterogeneous placement on each VM used. The new algorithm can find better combinations of map/reduce workers on multiple types of VMs than those homogeneous placement algorithms, thus saving more cost of using VMs than the homogeneous placement algorithms. Experimental results have shown that the operational cost of cloud-based MapReduce using the new placement algorithm was $24.3 - 44.0$ percent lower than that using the most popular homogeneous MapReduce placement algorithm.

The new placement algorithm also outperforms current heterogeneous placement algorithms. Unlike current heterogeneous placement algorithms, the new placement algorithm can make good use of the existing resources to further reduce the operational cost of cloud-based MapReduce. Experiments have been conducted to compare these two kinds of algorithms on saving the operational cost of cloud-based MapReduce. The experimental results have shown that the operational cost using the new algorithm was $2.0 - 36.2$ percent lower than that using the algorithm, without utilizing the spare resources.

In addition, the good scalability of the new placement algorithm has been demonstrated by case studies. The experimental results have shown that the computation times of the algorithm increased slowly when the problem size increased.

7.2.3 New MapReduce Consolidation Algorithm for Cloud-based MapReduce

A new MapReduce consolidation algorithm has been obtained, successfully solving the MRC problem. The MRC problem is a new type of consolidation problem, which has rarely been studied by current work. The MRC problem is much more complicated than most of the

consolidation problems studied in the current literature. Their consolidation problems only tackle homogeneous VMs or homogeneous MapReduce computations, whereas the MRC problem considers heterogeneous VMs and heterogeneous MapReduce computations. Few current consolidation algorithms can solve the MRC problem effectively.

Using the new MapReduce consolidation algorithm, the operational cost of cloud-based MapReduce has been greatly reduced. The experimental results have shown that using the new algorithm the cost of using VMs is $60.0 - 79.2$ percent less than that without MapReduce consolidation. Meanwhile, the solution found by the new algorithm shows good performance. Compared with an IFFD-based consolidation algorithm, the new algorithm saves $6.1 - 18.1$ percent more cost of using VMs and $14.3 - 49.7$ percent more migration cost. In addition, the experimental results have shown the good scalability of the new algorithm.

7.2.4 Novel Resource Provisioning Framework for Cloud-based MapReduce

A novel resource provisioning framework for cloud-based MapReduce has been developed, successfully solving the QoS-guaranteed resource provisioning problem for cloud-based MapReduce in dynamic environments. This resource provisioning framework is an event-driven one, totally different from current popular frameworks including the static provisioning frameworks and the periodical resource provisioning frameworks.

Unlike the static provisioning frameworks which provide fixed amount of resources, the new framework can scale up or scale down resource provisioning at runtime when initial resource provisioning is insufficient or over sufficient. Such a mechanism of the new framework greatly enhances the ability of QoS-guarantee while potentially reducing the operational cost of cloud-based MapReduce. The experimental results have shown that, using the static frameworks cannot guarantee QoS all the time whereas the new framework always guarantees QoS no matter how the environment changes. Also, the experimental results have shown that, using the new framework, the operational cost of cloud-based MapReduce is $15.3 - 27.1$ percent less than that using the static frameworks.

Also, unlike the periodical provisioning frameworks which provide the resource periodically, the new framework triggers resource provisioning only when it detects an event. Such event-driven mechanism of the new framework improves the accuracy of the amount and timing of resource provisioning, and also avoids the difficult issue of determining periods, as existed in

the periodical provisioning frameworks. Applying the event-driven mechanism, the new framework outperforms the periodical resource provisioning frameworks in the QoS guarantee and in the operational cost saving of cloud-based MapReduce. The experimental results have shown that the periodical resource provisioning frameworks cannot guarantee QoS when the period is long, whereas the new framework always guarantees QoS. Meanwhile, the experimental results have shown that using the new framework, the operational cost of cloud-based MapReduce is $2.5 - 15.8$ percent less than that using the periodical ones.

7.3 Future Work

This thesis has motivated some future work. One work is how to quantify the sufficient and necessary conditions for QoS-guaranteed resource provisioning on amount and timing. Current work has considered only the sufficient conditions, such that our work reserves only those conservative solutions for guaranteeing the QoS of cloud-based MapReduce, which potentially increases the operational cost of cloud-based MapReduce. Through characterizing the sufficient and necessary conditions, a more accurate resource provisioning solution could be obtained, and the operational cost of cloud-based MapReduce could be further reduced.

Another work is how to deploy the new resource provisioning framework on real cloud systems. Current work has evaluated the performance of the new framework by simulation. To widen the application scope of the new framework, a real QoS-guaranteed resource provisioning system based on our new framework will be developed in a real cloud environment.

References

AbdelBaky, M., Kim, H., Rodero, I., and Parashar, M. (2012). Accelerating MapReduce analytics using CometCloud. In *Proceedings of IEEE 5th International Conference on Cloud Computing (CLOUD)*, pages 447–454.

Ai, L., Tang, M., and Fidge, C. J. (2011). Resource allocation and scheduling of multiple composite web services in cloud computing using cooperative coevolution genetic algorithm. In *Proceedings of 18th International Conference on Neural Information Processing*, pages 258–267.

Alrokayan, M., Vahid Dastjerdi, A., and Buyya, R. (2014). Sla-aware provisioning and scheduling of cloud resources for big data analytics. In *Proceedings of 2014 IEEE International Conference on Cloud Computing in Emerging Markets (CCEM)*, pages 1–8.

Amazon (2015). Amazon EC2 instances page. [Online]. Available: http://aws.amazon.com/ec2/instance-types/.

Bang-Jensen, J. and Larsen, R. (2012). Efficient algorithms for real-life instances of the variable size bin packing problem. *Computers and Operations Research*, 39(11):2848 – 2857.

Beloglazov, A., Abawajy, J., and Buyya, R. (2012). Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. *Future Generation Computer Systems*, 28(5):755 – 768. Special Section: Energy efficiency in large-scale distributed systems.

Beloglazov, A. and Buyya, R. (2012). Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers. *Concurrency and Computation: Practice and Experience*, 24(13):1397–1420.

Beloglazov, A. and Buyya, R. (2013). Managing overloaded hosts for dynamic consolidation of virtual machines in cloud data centers under quality of service constraints. *IEEE Transactions on Parallel and Distributed Systems*, 24(7):1366–1379.

Bobroff, N., Kochut, A., and Beaty, K. (2007). Dynamic placement of virtual machines for managing sla violations. In *Proceedings of 10th IFIP/IEEE International Symposium on Integrated Network Management*, pages 119–128.

Brugger, B., Doerner, K., Hartl, R., and Reimann, M. (2004). Antpacking - an ant colony optimization approach for the one-dimensional bin packing problem. In *Evolutionary Computation in Combinatorial Optimization*, volume 3004 of *Lecture Notes in Computer Science*, pages 41–50. Springer Berlin Heidelberg.

Buyya, R., Yeo, C. S., Venugopal, S., Broberg, J., and Brandic, I. (2009). Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation Computer Systems*, 25(6):599 – 616.

Byun, E.-K., Kee, Y.-S., Kim, J.-S., and Maeng, S. (2011). Cost optimized provisioning of elastic resources for application workflows. *Future Generation Computer Systems*, 27(8):1011 – 1026.

Caprara, A. and Toth, P. (2001). Lower bounds and algorithms for the 2-dimensional vector packing problem. *Discrete Applied Mathematics*, 111(3):231–262.

Cardosa, M., Narang, P., Chandra, A., Pucha, H., and Singh, A. (2011). Steamengine: Driving MapReduce provisioning in the cloud. In *Proceedings of 18th International Conference on High Performance Computing (HiPC)*, pages 1–10.

Cardosa, M., Singh, A., Pucha, H., and Chandra, A. (2012). Exploiting spatio-temporal tradeoffs for energy-aware MapReduce in the cloud. *IEEE Transactions on Computers*, 61(12):1737–1751.

Chen, C. H., Lin, J. W., and Kuo, S. Y. (2014a). Deadline-constrained MapReduce scheduling based on graph modelling. In *Proceedings of 2014 IEEE 7th International Conference on Cloud Computing (CLOUD)*, pages 416–423.

Chen, K., Powers, J., Guo, S., and Tian, F. (2014b). Cresp: Towards optimal resource provisioning for MapReduce computing in public clouds. *IEEE Transactions on Parallel and Distributed Systems*, 25(6):1403–1412.

Chen, Z. G., Du, K. J., Zhan, Z. H., and Zhang, J. (2015). Deadline constrained cloud computing resources scheduling for cost optimization based on dynamic objective genetic algorithm. In *Proceedings of 2015 IEEE Congress on Evolutionary Computation (CEC)*, pages 708–714.

Cheng, D., Rao, J., Jiang, C., and Zhou, X. (2015). Resource and deadline-aware job scheduling in dynamic hadoop clusters. In *Parallel and Distributed Processing Symposium (IPDPS), 2015 IEEE International*, pages 956–965.

Chu, C., Kim, S. K., Lin, Y.-A., Yu, Y., Bradski, G., Ng, A. Y., and Olukotun, K. (2007). Map-Reduce for machine learning on multicore. *Advances in Neural Information Processing Systems*, 19:281–288.

CPLEX (2015). IBM CPLEX Optimizer. [Online]. Available: http://ibm.com/software/commerce/optimization/cplex-optimizer/.

Crainic, T. G., Perboli, G., Rei, W., and Tadei, R. (2011). Efficient lower bounds and heuristics for the variable cost and size bin packing problem. *Computers and Operations Research*, 38(11):1474 – 1482.

Daryani, M. H. and Thakare, M. S. B. (2015). Deadline and cost based MapReduce job scheduling in heterogeneous cloud using dynamic pricing. *International Journal*, 3(9):56–64.

Dean, J. and Ghemawat, S. (2008). MapReduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113.

Dong, X., Wang, Y., and Liao, H. (2011). Scheduling mixed real-time and non-real-time applications in MapReduce environment. In *Proceedings of 2011 IEEE 17th International Conference on Parallel and Distributed Systems (ICPADS)*, pages 9–16.

Dyckhoff, H. (1990). A typology of cutting and packing problems. *European Journal of Operational Research*, 44(2):145 – 159. Cutting and Packing.

Falkenauer, E. (1996). A hybrid grouping genetic algorithm for bin packing. *Journal of Heuristics*, 2(1):5–30.

Farley, B., Juels, A., Varadarajan, V., Ristenpart, T., Bowers, K. D., and Swift, M. M. (2012). More for your money: exploiting performance heterogeneity in public clouds. In *Proceedings of the 3rd ACM Symposium on Cloud Computing*, page 20. ACM.

Ferreto, T. C., Netto, M. A., Calheiros, R. N., and Rose, C. A. D. (2011). Server consolidation with migration control for virtualized data centers. *Future Generation Computer Systems*, 27(8):1027 – 1034.

Ganapathi, A., Chen, Y., Fox, A., Katz, R., and Patterson, D. (2010). Statistics-driven workload modeling for the cloud. In *Proceedings of 2010 IEEE 26th International Conference on Data Engineering Workshops (ICDEW)*, pages 87–92.

Gandhi, A., Dube, P., Kochut, A., and Zhang, L. (2015). Model-driven autoscaling for hadoop clusters. In *Proceedings of 2015 IEEE International Conference on Autonomic Computing (ICAC)*, pages 155–156.

Ganglia (2015). Ganglia monitoring system. [Online]. Available: http://ganglia.sourceforge .net/.

García-Valls, M., Cucinotta, T., and Lu, C. (2014). Challenges in real-time virtualization and predictable cloud computing. *Journal of Systems Architecture*, 60(9):726–740.

Garfinkel, R. S. and Nemhauser, G. L. (1972). *Integer programming*, volume 4. Wiley New York.

Garg, S. K., Gopalaiyengar, S. K., and Buyya, R. (2011). *Algorithms and Architectures for Parallel Processing: 11th International Conference, ICA3PP, Melbourne, Australia, October 24-26, 2011, Proceedings, Part I*, chapter SLA-Based Resource Provisioning for Heterogeneous Workloads in a Virtualized Cloud Datacenter, pages 371–384. Springer Berlin Heidelberg, Berlin, Heidelberg.

Gunarathne, T., Wu, T.-L., Qiu, J., and Fox, G. (2010). MapReduce in the clouds for science. In *Proceedings of IEEE 2nd International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 565–572.

Hadoop (2015). Hadoop releases. [Online]. Available: http://hadoop.apache.org/releases.html.

Haouari, M. and Serairi, M. (2009). Heuristics for the variable sized bin-packing problem. *Computers and Operations Research*, 36(10):2877–2884.

He, C., Weitzel, D., Swanson, D., and Lu, Y. (2012). Hog: Distributed hadoop MapReduce on the grid. In *Proceedings of 2012 SC Companion: High Performance Computing, Networking, Storage and Analysis (SCC),*, pages 1276–1283.

He, Q., Shang, T., Zhuang, F., and Shi, Z. (2013). Parallel extreme learning machine for regression based on MapReduce. *Neurocomputing*, 102(0):52 – 58.

Herodotou, H. and Babu, S. (2011). Profiling, what-if analysis, and cost-based optimization of MapReduce programs. *Proceedings of International Conference on VLDB Endowment*, 4(11):1111–1122.

Herodotou, H., Dong, F., and Babu, S. (2011). No one (cluster) size fits all: automatic cluster sizing for data-intensive analytics. In *Proceedings of ACM 2nd Symposium on Cloud Computing*, page 18.

Huang, S., Huang, J., Dai, J., Xie, T., and Huang, B. (2010). The hibench benchmark suite: Characterization of the MapReduce-based data analysis. In *Proceedings of IEEE 26th International Conference on on Data Engineering Workshops (ICDEW)*, pages 41–51.

Huang, Z., Tsang, D. H. K., and She, J. (2012). A virtual machine consolidation framework for MapReduce enabled computing clouds. In *Proceedings of 24th International Teletraffic Congress*, ITC '12, pages 26:1–26:8. International Teletraffic Congress.

Hwang, E. and Kim, K. H. (2012). Minimizing cost of virtual machines for deadline-constrained MapReduce applications in the cloud. In *Proceedings of ACM/IEEE 13th International Conference on Grid Computing (GRID)*, pages 130–138.

Iima, H. and Yakawa, T. (2003). A new design of genetic algorithm for bin packing. In *Proceedings of IEEE 2003 Congress on Evolutionary Computation*, pages 1044–1049.

Kang, J. and Park, S. (2003). Algorithms for the variable sized bin packing problem. *European Journal of Operational Research*, 147(2):365–372.

Kc, K. and Anyanwu, K. (2010). Scheduling hadoop jobs to meet deadlines. In *Proceedings of IEEE 2nd International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 388–392.

Kijsipongse, E. and U-ruekolan, S. (2014). Scaling hadoop clusters with virtualized volunteer computing environment. In *Proceedings of 2014 11th International Joint Conference onComputer Science and Software Engineering (JCSSE)*, pages 146–151.

Lama, P. and Zhou, X. (2012). AROMA: Automated resource allocation and configuration of MapReduce environment in the cloud. In *Proceedings of 9th International Conference on Autonomic Computing*, pages 63–72.

Lang, W. and Patel, J. M. (2010). Energy management for MapReduce clusters. *Proceedings of VLDB Endow.*, 3(1-2):129–139.

Lee, K. and Figueiredo, R. (2012). MapReduce on opportunistic resources leveraging resource availability. In *Proceedings of 2012 IEEE 4th International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 435–442.

Lee, S., Panigrahy, R., Prabhakaran, V., Ramasubramanian, V., Talwar, K., Uyeda, L., and Wieder, U. (2011). Validating heuristics for virtual machines consolidation. *Microsoft Research, MSR-TR-2011-9*.

Li, H., Wang, J., Peng, J., Wang, J., and Liu, T. (2013). Energy-aware scheduling scheme using workload-aware consolidation technique in cloud data centres. *Communications, China*, 10(12):114–124.

Li, H., Wei, X., Fu, Q., and Luo, Y. (2014a). MapReduce delay scheduling with deadline constraint. *Concurrency and Computation: Practice and Experience*, 26(3):766–778.

Li, H.-H., Chen, Z.-G., Zhan, Z.-H., Du, K.-J., and Zhang, J. (2015). Renumber coevolutionary multiswarm particle swarm optimization for multi-objective workflow scheduling on cloud computing environment. In *Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation*, GECCO Companion '15, pages 1419–1420. ACM.

Li, S., Hu, S., Wang, S., Su, L., Abdelzaher, T., Gupta, I., and Pace, R. (2014b). Woha: Deadline-aware map-reduce workflow scheduling framework over hadoop clusters. In *Proceedings of 2014 IEEE 34th International Conference on on Distributed Computing Systems (ICDCS)*, pages 93–103.

Lin, C.-C., Liu, P., and Wu, J.-J. (2011). Energy-aware virtual machine dynamic provision and scheduling for cloud computing. In *Proceedings of 2011 IEEE International Conference on Cloud Computing (CLOUD)*, pages 736–737.

Lin, H., Ma, X., Archuleta, J., Feng, W.-c., Gardner, M., and Zhang, Z. (2010a). Moon: MapReduce on opportunistic environments. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, HPDC '10, pages 95–106. ACM.

Lin, H., Ma, X., Archuleta, J., Feng, W.-c., Gardner, M., and Zhang, Z. (2010b). Moon: MapReduce on opportunistic environments. In *Proceedings of ACM 19th International Symposium on High Performance Distributed Computing*, pages 95–106.

Lingo (2015). Lindo system inc. . [Online]. Available: http://www.lindo.com/.

Liu, D., Tan, K., Huang, S., Goh, C., and Ho, W. (2008). On solving multiobjective bin packing problems using evolutionary particle swarm optimization. *European Journal of Operational Research*, 190(2):357 – 382.

Liu, L., Zhou, Y., Liu, M., Xu, G., Chen, X., Fan, D., and Wang, Q. (2012). Preemptive hadoop jobs scheduling under a deadline. In *Proceedings of 2012 Eighth International Conference on Semantics, Knowledge and Grids (SKG)*, pages 72–79.

Maheshwari, N., Nanduri, R., and Varma, V. (2012). Dynamic energy efficient data placement and cluster reconfiguration algorithm for MapReduce framework. *Future Generation Computer Systems*, 28(1):119 – 127.

Maiza, M., Labed, A., and Radjef, M. (2013). Efficient algorithms for the offline variable sized bin-packing problem. *Journal of Global Optimization*, 57(3):1025–1038.

Malekimajd, M., Ardagna, D., Ciavotta, M., Rizzi, A. M., and Passacantando, M. (2015). Optimal map reduce job capacity allocation in cloud systems. *SIGMETRICS Perform. Eval. Rev.*, 42(4):51–61.

Malekimajd, M., Rizzi, A., Ardagna, D., Ciavotta, M., Passacantando, M., and Movaghar, A. (2014). Optimal capacity allocation for executing MapReduce jobs in cloud systems. In *Proceedings of 2014 16th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*, pages 385–392.

Mao, M. and Humphrey, M. (2011). Auto-scaling to minimize cost and meet application deadlines in cloud workflows. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '11, pages 49:1–49:12. ACM.

Mattess, M., Calheiros, R., and Buyya, R. (2013). Scaling MapReduce applications across hybrid clouds to meet soft deadlines. In *Proceedings of IEEE 27th International Conference on Advanced Information Networking and Applications (AINA)*, pages 629–636.

Meng, Z., Li, J., Zhou, Y., Liu, Q., Liu, Y., and Cao, W. (2011). bCloudBLAST: An efficient MapReduce program for bioinformatics applications. In *Proceedings of IEEE 4th International Conference on Biomedical Engineering and Informatics (BMEI)*, volume 4, pages 2072–2076. IEEE.

Monaci, M. and Toth, P. (2006). A set-covering-based heuristic approach for bin-packing problems. *INFORMS Journal on Computing*, 18(1):71–85.

Numoto, T. (2015). Microsoft announces Azure IoT suite. [Online]. Available: https://blogs.microsoft.com/iot/2015/03/16/microsoft-announces-azure-iot-suite/.

Palanisamy, B., Singh, A., and Liu, L. (2014). Cost-effective resource provisioning for MapReduce in a cloud. *IEEE Transactions on Parallel and Distributed Systems*, PP(99):1–1.

Palanisamy, B., Singh, A., Liu, L., and Jain, B. (2011). Purlieus: locality-aware resource allocation for MapReduce in a cloud. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, number 58, pages 1–11.

Panigrahy, R., Talwar, K., Uyeda, L., and Wieder, U. (2011). Heuristics for vector bin packing. *http://research. microsoft. com*.

Petrucci, V., Carrera, E. V., Loques, O., Leite, J. C., and Mosse, D. (2011). Optimized management of power and performance for virtualized heterogeneous server clusters. In *Proceedings of 2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pages 23–32.

Polo, J., Becerra, Y., Carrera, D., Steinder, M., Whalley, I., Torres, J., and Ayguade, E. (2013). Deadline-based MapReduce workload management. *IEEE Transactions on Network and Service Management*, 10(2):231–244.

Polo, J., Carrera, D., Becerra, Y., and Steinder, M. (2010). Performance-driven task co-scheduling for MapReduce environments. In *Proceedings of 2010 IEEE Symposium on Network Operations and Management Symposium (NOMS)*, pages 373–380.

Polo, J., Castillo, C., Carrera, D., Becerra, Y., Whalley, I., Steinder, M., Torres, J., and Ayguad, E. (2011). Resource-aware adaptive scheduling for MapReduce clusters. In *Middleware 2011*, volume 7049 of *Lecture Notes in Computer Science*, pages 187–207. Springer Berlin Heidelberg.

Rao, B. T. and Reddy, L. S. S. (2012). Scheduling data intensive workloads through virtualization on MapReduce based clouds. *International Journal of Distributed and Parallel Systems*, 3(4):99–110.

Ruiz-Alvarez, A., Kim, I. K., and Humphrey, M. (2015). Toward optimal resource provisioning for cloud MapReduce and hybrid cloud applications. In *Proceedings of 2015 IEEE 8th International Conference on Cloud Computing (CLOUD)*, pages 669–677.

Sampaio, A. M. and Barbosa, J. G. (2016). Chapter three - energy-efficient and sla-based resource management in cloud data centers. In Hurson, A. R. and Sarbazi-Azad, H., editors, *Energy Efficiency in Data Centers and Clouds*, volume 100 of *Advances in Computers*, pages 103 – 159. Elsevier.

Schad, J., Dittrich, J., and Quiané-Ruiz, J.-A. (2010). Runtime measurements in the cloud: observing, analyzing, and reducing variance. *Proceedings of VLDB Endowment*, 3(1-2):460–471.

Segura, C., Segredo, E., and León, C. (2011). Parallel island-based multiobjectivised memetic algorithms for a 2d packing problem. In *Proceedings of ACM 13th Annual Conference on Genetic and Evolutionary Computation*, GECCO '11, pages 1611–1618.

Shi, W. and Hong, B. (2013). Clotho: An elastic MapReduce workload/runtime co-design. In *Proceedings of 12th International Workshop Adaptive and Reflective Middleware*, pages 1–6.

Singh, S. and Chana, I. (2015). Q-aware: Quality of service based cloud resource provisioning. *Computers & Electrical Engineering*, 47:138 – 160.

Srikantaiah, S., Kansal, A., and Zhao, F. (2008). Energy aware consolidation for cloud computing. In *Proceedings of 2008 Conference on Power Aware Computing and Systems*, volume 10.

Tang, B., Moca, M., Chevalier, S., He, H., and Fedak, G. (2010). Towards MapReduce for desktop grid computing. In *Proceedings of 2010 International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC)*, pages 193–200.

Tang, Z., Zhou, J., Li, K., and Li, R. (2012). A MapReduce task scheduling algorithm for deadline constraints. *Cluster Computing*, 16(4):651–662.

Tavangar, J. (2014). Large-scale hadoop installations are the new norm. [Online]. Available: http://www.thearmadagroup.com/it-infrastructure/large-scale-hadoop-installations-are-the-new-norm.

Taylor, R. C. (2010). An overview of the Hadoop/MapReduce/HBase framework and its current applications in bioinformatics. *BMC bioinformatics*, 11(Suppl 12):S1.

Teng, F., Magoulès, F., Yu, L., and Li, T. (2014). A novel real-time scheduling algorithm and performance analysis of a MapReduce-based cloud. *The Journal of Supercomputing*, 69(2):739–765.

Teng, F., Yang, H., Li, T., Magoulès, F., and Fan, X. (2015). Mus: a novel deadline-constrained scheduling algorithm for hadoop. *International Journal of Computational Science and Engineering*, 11(4):360–367.

Tian, F. and Chen, K. (2011). Towards optimal resource provisioning for running MapReduce programs in public clouds. In *Proceedings of IEEE International Conference on Cloud Computing (CLOUD)*, pages 155–162.

Verma, A., Ahuja, P., and Neogi, A. (2008). pmapper: power and migration cost aware application placement in virtualized systems. In *Middleware 2008*, pages 243–264. Springer.

Verma, A., Cherkasova, L., and Campbell, R. (2011a). Resource provisioning framework for MapReduce jobs with performance goals. In *Middleware 2011*, volume 7049 of *Lecture Notes in Computer Science*, pages 165–186. Springer Berlin Heidelberg.

Verma, A., Cherkasova, L., and Campbell, R. H. (2011b). ARIA: Automatic resource inference and allocation for MapReduce environments. In *Proceedings of ACM 8th International Conference on Autonomic Computing*, pages 235–244.

Versace, C. (2014). Talking big data and analytics with IBM. [Online]. Available: http://www.forbes.com/sites/chrisversace/2014/04/01/talking-big-data-and-analytics-with-ibm/.

VMware (2015). VMware homepage. [Online]. Available: http://www.vmware.com.

Wang, J., Li, Q., and Shi, Y. (2013). SLO-driven task scheduling in MapReduce environments. In *Proceedings of 10th Conference on Web Information System and Application (WISA)*, pages 308–313.

Wang, J. and Li, X. (2015). Task scheduling for MapReduce in heterogeneous networks. *Cluster Computing*, pages 1–14.

Wang, K., Tan, B., Shi, J., and Yang, B. (2011). Automatic task slots assignment in hadoop MapReduce. In *Proceedings of 1st Workshop Architectures and Systems for Big Data*, ASBD '11, pages 24–29.

Wang, X., Shen, D., Bai, M., Nie, T., Kou, Y., and Yu, G. (2014). Sames: deadline-constraint scheduling in MapReduce. *Frontiers of Computer Science*, 9(1):128–141.

White, B., Yeh, T., Lin, J., and Davis, L. (2010). Web-scale computer vision using MapReduce for multimedia data mining. In *Proceedings of 10th International Workshop on Multimedia Data Mining*, MDMKDD '10, pages 9:1–9:10.

White, T. (2009). *Hadoop: The Definitive Guide*. O'Reilly Media.

Wilcox, D., McNabb, A., and Seppi, K. (2011). Solving virtual machine packing with a reordering grouping genetic algorithm. In *Proceedings of IEEE 2011 Congress on Evolutionary Computation*, pages 362–369.

Wolf, J., Rajan, D., Hildrum, K., Khandekar, R., Kumar, V., Parekh, S., Wu, K.-L., and Balmin, A. (2010). Flex: A slot allocation scheduling optimizer for MapReduce workloads. In Gupta, I. and Mascolo, C., editors, *Middleware 2010*, volume 6452 of *Lecture Notes in Computer Science*, pages 1–20. Springer Berlin Heidelberg.

Wolsey, L. A. (2008). *Mixed Integer Programming*. Wiley Encyclopedia of Computer Science and Engineering.

Wu, G., Tang, M., Tian, Y., and Li, W. (2012). Energy-efficient virtual machine placement in data centers by genetic algorithm. In *Proceedings of 19th International Conference on Neural Information Processing*, pages 315–323.

Wu, X., Zhu, X., Wu, G.-Q., and Ding, W. (2014). Data mining with big data. *IEEE Transactions on Knowledge and Data Engineering*, 26(1):97–107.

Xiang, Y., Balasubramanian, B., Wang, M., Lan, T., Sen, S., and Chiang, M. (2013). Self-adaptive, deadline-aware resource control in cloud computing. In *Proceedings of 2013 IEEE 7th International Conference on Self-Adaptation and Self-Organizing Systems Workshops (SASOW)*, pages 41–46.

Xiao, Z., Song, W., and Chen, Q. (2013). Dynamic resource allocation using virtual machines for cloud computing environment. *IEEE Transactions on Parallel and Distributed Systems*, 24(6):1107–1117.

Xie, J., Yin, S., Ruan, X., Ding, Z., Tian, Y., Majors, J., Manzanares, A., and Qin, X. (2010). Improving MapReduce performance through data placement in heterogeneous hadoop clusters. In *Proceedings of 2010 IEEE International Symposium on Parallel and Distributed Processing, Workshops and Phd Forum (IPDPSW)*, pages 1–9.

Xu, X. and Tang, M. (2014a). A more efficient and effective heuristic algorithm for the MapReduce placement problem in cloud computing. In *Proceedings of IEEE 7th International Conference on Cloud Computing (CLOUD)*, pages 264–271.

Xu, X. and Tang, M. (2014b). A new grouping genetic algorithm for the MapReduce placement problem in cloud computing. In *Proceedings of 2014 IEEE Congress on Evolutionary Computation (CEC)*, pages 1601–1608.

Xu, X., Tang, M., and Tian, Y. (2015). A new approach to the cloud-based heterogeneous MapReduce placement problem. *IEEE Transactions on Services Computing*, accepted on 15th, May, 2015, in press.

Xu, X., Tang, M., and Tian, Y. (2016). Theoretical results of QoS-guaranteed resource scaling for cloud-based MapReduce. *IEEE Transactions on Cloud Computing*, accepted on 18th, February, 2016, in press.

Zaharia, M., Borthakur, D., Sarma, J. S., Elmeleegy, K., Shenker, S., and Stoica, I. (2009). Job scheduling for multi-user MapReduce clusters. *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2009-55*.

Zaharia, M., Konwinski, A., Joseph, A. D., Katz, R. H., and Stoica, I. (2008). Improving MapReduce performance in heterogeneous environments. In *OSDI*, volume 8, page 7.

Zhan, Z.-H., Liu, X.-F., Gong, Y.-J., Zhang, J., Chung, H. S.-H., and Li, Y. (2015). Cloud computing resource scheduling and a survey of its evolutionary approaches. *ACM Comput. Surv.*, 47(4):63:1–63:33.

Zhang, Q., Zhani, M., Boutaba, R., and Hellerstein, J. (2014a). Dynamic heterogeneity-aware resource provisioning in the cloud. *IEEE Transactions on Cloud Computing*, 2(1):14–28.

Zhang, W., Rajasekaran, S., Wood, T., and Zhu, M. (2014b). Mimp: Deadline and interference aware scheduling of hadoop virtual machines. In *Proceedings of 2014 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pages 394–403.

Zhang, X., Ju, S., and Jiao, Z. (2012). A scheduling method based on deadlines in MapReduce. In *Electrical, Information Engineering and Mechatronics 2011*, pages 1585–1592. Springer.

Zhang, Z., Cherkasova, L., and Loo, B. T. (2015). Exploiting cloud heterogeneity to optimize performance and cost of MapReduce processing. *SIGMETRICS Perform. Eval. Rev.*, 42(4):38–50.

Zhou, A. C. and He, B. (2014). Transformation-based monetary cost optimizations for workflows in the cloud. *IEEE Transactions on Cloud Computing*, 2(1):85–98.