Cost-effective and Qos-aware resource allocation for cloud computing

Wei, Lei

2016

Wei, L. (2016). Cost-effective and Qos-aware resource allocation for cloud computing.
Doctoral thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/66012>

<https://doi.org/10.32657/10356/66012>

**COST-EFFECTIVE AND QOS-AWARE
RESOURCE ALLOCATION FOR CLOUD
COMPUTING**

WEI LEI

School of Computer Engineering

A thesis submitted to the Nanyang Technological University
in fulfillment of the requirement for the degree of
Doctor of Philosophy

2015

Abstract

As the most important problem in cloud computing technology, resource allocation not only affects the cost of the cloud operators and users, but also impacts the performance of cloud jobs. Provisioning too much resource in clouds wastes energy and cost while provisioning too few resource will cause performance degradation of cloud applications. Current researches in the resource allocation field mainly focus on homogeneous resource allocation and take CPU as the most important resource in resource allocation. However, as resource demands of cloud workloads get increasingly heterogeneous on different resource types, current methods are not suitable for some other type of jobs such as memory-intensive applications. They are neither efficient in terms of offering economical and high-quality resource allocation in clouds.

In this thesis, we firstly propose a resource provisioning method, namely BigMem, to consider the features of resource allocation based on memory. Memory-intensive applications have recently become popular for high-throughput and low-latency computing. Current resource provisioning methods focus more on other resources such as CPU and network bandwidth which are considered as the bottlenecks in traditional cloud applications. However, for memory-intensive jobs, main memories are always the bottleneck resource for performance. Therefore, main memory should be the first consideration in resource allocation and provisioning for VMs in clouds hosting memory-intensive applications. By considering the unique behavior of resource provisioning for memory-intensive jobs, BigMem is able to effectively reduce the resource usage for dynamic workloads in clouds. Specifically, we seek Markov Chain modeling to periodically determine the required number of PMs and further optimize the resource utilization by conducting VM migration and resource overcommit. We evaluate our design using simulation with synthetic and real world traces. Experiments results show that BigMem is able to provision

the appropriate number of resources for highly dynamic workloads while keeping an acceptable service-level-agreement (SLA). By comparisons, BigMem reduces the average number of active machines in data center by 63% and 27% on average than peak-load provisioning and heuristic methods, respectively. These results translate into good performance for users and low cost for cloud providers.

To support different types of workloads in clouds (such as memory-intensive and computation-intensive applications), we then propose a heterogeneous resource allocation method, skewness-avoidance multi-resource allocation (SAMR), that considers the skewness of different resource types to optimize the resource usage in clouds. Current IaaS clouds provision resources in terms of virtual machines (VMs) with homogeneous resource configurations where different types of resources in VMs have similar share of the capacity in a physical machine (PM). However, most user jobs demand different amounts for different resources. For instance, high-performance-computing jobs require more CPU cores while memory-intensive applications require more memory. The existing homogeneous resource allocation mechanisms cause resource starvation where dominant resources are starved while non-dominant resources are wasted. To overcome this issue, we propose SAMR to allocate resource according to diversified requirements on different types of resources. Our solution includes a job allocation algorithm to ensure heterogeneous workloads are allocated appropriately to avoid skewed resource utilization in PMs, and a model-based approach to estimate the appropriate number of active PMs to operate SAMR. We show relatively low complexity for our model-based approach for practical operation and accurate estimation. Extensive simulation results show the effectiveness of SAMR and the performance advantages over its counterparts.

Finally, we turn to a resource allocation problem in a specific application for media computing in clouds. As the “biggest big data”, video data streaming in the network contributes the largest portion of global traffic nowadays and in future. Due to heterogeneous mobile devices, networks and user preferences, the demands of transcoding source videos into different versions have been increased significantly. However, video transcoding is a time-consuming task and how to guarantee quality-of-service (QoS) for large video data is very challenging, particularly for those real-time applications which hold strict delay requirement. In this thesis, we propose a cloud-based online video

transcoding system (COVT) aiming to offer economical and QoS guaranteed solution for online large-volume video transcoding. COVT utilizes performance profiling technique to obtain different performance of transcoding tasks in different infrastructures. Based on the profiles, we model the cloud-based transcoding system as a queue and derive the QoS values of the system based on queuing theory. With the analytically derived relationship between QoS values and the number of CPU cores required for transcoding workloads, COVT is able to solve the optimization problem and obtain the minimum resource reservation for specific QoS constraints. A task scheduling algorithm is further developed to dynamically adjust the resource reservation and schedule the tasks so as to guarantee the QoS. We implement a prototype system of COVT and experimentally study the performance on real-world workloads. Experimental results show that COVT effectively provisions minimum number of resources for predefined QoS. To validate the effectiveness of our proposed method under large scale video data, we further perform simulation evaluation which again shows that COVT is capable to achieve cost-effective and QoS-aware video transcoding in cloud environment.

Acknowledgments

I would like to give my sincere acknowledgement to my previous supervisor Dr. Foh Chuan Heng, my current supervisor Dr. Cai Jianfei and my co-supervisor Dr. He Bingsheng for dedicating their knowledge, encouragement and support in guidance of my research work.

I would also like to thank the members of my PhD dissertation examination committee for their valuable time and advice.

Finally, I would express my wholehearted gratitude towards my families and my friends for their dedication and love throughout my life.

Contents

Abstract	ii
Acknowledgments	v
List of Figures	ix
List of Tables	xi
1 Introduction	1
1.1 Background	1
1.2 Research Challenges and Objective	3
1.3 Thesis Contributions	5
1.4 Thesis Organization	6
2 Literature Review	8
2.1 Homogeneous Resource Allocation	8
2.2 Heterogeneous Resource Allocation	10
2.3 Cloud-based Video Transcoding	11
3 Efficient Resource Management for Memory-Intensive Applications in Clouds	14
3.1 Introduction	14
3.2 Big Memory Clouds	16

3.3	System Overview	19
3.4	System Modeling	25
3.4.1	The Base model	25
3.4.2	Migration Overhead	29
3.4.3	Overcommit	30
3.4.4	Complexity Analysis	32
3.5	Evaluation	32
3.5.1	Methodology	33
3.5.2	Overall Results	35
3.5.3	Sensitivity Studies	40
3.6	Conclusion	41
4	Efficient Resource Allocation for Heterogeneous Workloads in IaaS Clouds	43
4.1	Introduction	43
4.2	System overview	47
4.3	Skewness-Avoidance Multi-Resource allocation	50
4.3.1	New Notions of VM Offering	50
4.3.2	Multi-Resource Skewness	51
4.3.3	Skewness-Avoidance Resource allocation	53
4.4	Resource Prediction Model	55
4.5	Evaluation	60
4.5.1	Experimental setup	61
4.5.2	Experimental results	63
4.6	Conclusion	69

5	QoS-aware Resource Allocation for Video Transcoding in Clouds	71
5.1	Introduction	71
5.2	System architecture	74
5.2.1	Video consumer	75
5.2.2	Video service provider	76
5.2.3	Cloud cluster	78
5.3	Performance Profiling	79
5.4	Resource Prediction Model	80
5.4.1	Queuing model	80
5.4.2	Solution	82
5.5	Task scheduling	86
5.6	Testbed Experiments	90
5.6.1	Experiment setup	90
5.6.2	Experimental Results	92
5.7	Simulation Evaluation	96
5.8	Conclusion	98
6	Conclusion and Future Works	100
6.1	Conclusion	100
6.2	Future Directions	102
6.2.1	Extension of Resource Allocation in Clouds	102
6.2.2	Other Research Issues in Cloud Computing	103
	References	106

List of Figures

1.1	The cases of over-provisioning, under-provisioning and delay caused by under-provisioning.	3
3.1	Experiments results of memcached.	17
3.2	Flowchart of BigMem algorithm.	21
3.3	The Base Model for BigMem with a FF scheduling policy.	27
3.4	The six workloads with time (hour)	34
3.5	Number of active PMs in each time slot.	37
3.6	The overflow probability of four synthetic workloads and two real workloads.	38
3.7	The results of four components of BigMem for four synthetic workloads. .	39
3.8	Sensitivity studies for workload Stable.	40
4.1	Resource usage analysis of Google Cluster Traces.	45
4.2	System architecture of SAMR.	48
4.3	State transitions in the model.	57
4.4	Three synthetic workload patterns and one real world cloud trace from Google.	63
4.5	Overall results of four metrics under four workloads. The bars in the figure show average values and the red lines indicate 95% confidence intervals. .	64
4.6	Detailed results of three metrics under four workload patterns.	66

4.7	Sensitivity studies for different degrees of heterogeneity (job distributions). The bars in the figure show average values and the red lines indicate 95% confidence intervals.	67
4.8	Sensitivity studies for delay threshold, maximum VM capacity and length of time slot using Google trace.	68
5.1	System architecture of COVT.	75
5.2	The relationship between transcoding modes and QoS.	80
5.3	The queuing model in COVT.	81
5.4	Processing time of tasks.	84
5.5	Illustration of video transcoding tasks scheduling.	88
5.6	Workloads in experiments.	91
5.7	Profiling results with two transcoding modes and four video types.	93
5.8	Comparison of resource provisioning for different methods.	94
5.9	Detailed results of slow mode proportion, delay and chunk size.	94
5.10	Parameter studies of the testbed experiments.	95
5.11	Simulation results for large scale data set.	96
5.12	Number of CPU cores per stream.	97

List of Tables

3.1	Notations of BigMem algorithm	20
3.2	Overall results in total machine hours	35
4.1	Notations used in algorithms and models	47
5.1	Notations used in the transcoding system	74

Chapter 1

Introduction

1.1 Background

Public clouds have attracted much attention from both industry and academia recently. Users are able to benefit from clouds by highly elastic, scalable and economical resource utilizations. By using public clouds, users no longer need to purchase and maintain sophisticated hardware, which can be translated to simpler system maintenance and lower investments. Generally, cloud computing technology is divided into three categories, namely infrastructure-as-a-service (IaaS), platform-as-a-service (PaaS) and software-as-a-service (SaaS), according to the manner of resource provisioning and the targeted applications. IaaS clouds provide flexible computing ability by provisioning resources in terms of virtual machines (VMs) which consist of different kinds of physical resources. Users are able to access physical resource and run their applications in IaaS clouds (e.g., Amazon EC2 [1]). PaaS clouds aim at providing more powerful components for some specific applications such as web development (e.g., Google App Engine). SaaS clouds target at providing applications to end users directly (e.g., Dropbox). All three categories of clouds adopt a pay-as-you-go manner to charge users according to the leased resource amounts (in IaaS clouds) or service subscription amounts (in PaaS or SaaS clouds).

Resource allocation is the most important issue in operating cloud systems especially in IaaS clouds where physical resources are directly provisioned and charged. In the perspective of cloud operators (or providers), on one hand, provisioning too many resources will cause unnecessary energy consumption which increases the costs. On the other hand, provisioning too few resources causes poor performance in terms of delaying hosting users' VM requests. Similarly, the resource allocation problem is also important from the view of cloud users. Leasing too many resources from clouds wastes money but reserving too few resources may cause performance degradation of users' jobs.

It is clear that there is a trade-off in resource allocation between low cost and high quality-of-service (QoS) in clouds. To achieve cost-effective cloud computing, we expect to use as few resource as possible to complete the considered jobs. However, the QoS or performance of these jobs will be low if too few resource is provisioned because the workloads may be too big for provisioned resource to do the computation. However, it is difficult to get the optimal solution in a practical system under the trade-off between cost and QoS because of the dynamic workloads in clouds. This thesis targets at addressing the trade-off between cost and QoS in resource allocation in cloud computing environment, which is detailed introduced in Section 1.2.

Current research works in the field of resource allocation in clouds mainly consider a homogeneous method where CPU is considered as the dominant resource. Thus, it lacks support for both cost-effective and QoS-aware property for many other application scenarios. This thesis studies cost-effective and QoS-aware resource allocation methods for three specific application scenarios in order to improve cost consumption with QoS constraints. In the next section, we introduce the research problems and the challenges as well as the motivations for the three research problems.

Figure 1.1: The cases of over-provisioning, under-provisioning and delay caused by under-provisioning.

1.2 Research Challenges and Objective

As discussed, there is a trade-off between cost and QoS in clouds for all kinds of workloads. We require to design sophisticated resource provisioning and scheduling algorithms to keep low cost and acceptable QoS. However, it is challenging to provision and schedule resource in clouds that consists of a large amount of physical machines (PMs). Since workloads always vary against time in clouds, a fixed resource provisioning and scheduling method is obviously inefficient as shown in Fig. 1.1. The case of over-provisioning wastes resource while the case of under-provisioning causes QoS degradation in terms of delayed job completion. Thus, it is necessary to design a dynamic resource management scheme that provisions appropriate number of resources for the certain amount of workloads in every small time slot.

Nevertheless, estimating appropriate resource amounts for cloud workloads is also a non-trivial task. Firstly, there are a number of random variables in cloud systems that complicate the prediction of resource usages including the workload amounts (e.g., VM requests in IaaS clouds, transcoding requests in cloud-based video transcoding system, etc), requests arrival times and execution time of jobs. Secondly, the solution space will be very huge because of large amount of PMs involved. This issue becomes more complex after heterogenous resource types are considered. Thirdly, different QoS requirements of

applications are hard to be satisfied unless the relationship between QoS and resource amounts are derived. Moreover, how to compensate the mismatch between prediction and practical QoS in the runtime is also a big question.

To address above challenges, we studied three research problems of resource allocation in three different scenarios in this thesis. After the literature review, we noticed that current resource management mainly focuses on the scenario where CPU is the dominant resource, which is inefficient for some other type of applications such as in-memory storage or processing. Thus, we firstly propose a novel resource provisioning method, namely BigMem, to optimize the resource usage and QoS in big memory clouds. This work is the first attempt to design a resource management method considering unique features of memory-intensive applications. Since the first work considers a different scenario from previous works focusing on CPU-intensive applications, we naturally try to address the problem of resource allocation for heterogeneous workloads with both CPU- and memory-intensive applications in clouds as our second research problem. The second work considers CPU- and memory-intensive jobs together and schedule them with a skewness-avoidance algorithm to reduce the unbalance of resource usages among multiple resource types. Lastly, we study a resource allocation problem for a specific application in media computing in the third work after we addressed two resource management problems for general workloads. The considered application scenario is cloud-based video transcoding which transcodes a large number of video streams generated from source in real-time.

The objective of these three works is to reduce the resource reservation in clouds while satisfying acceptable QoS of cloud jobs. We make effort to find an optimal solution that minimizes the amount of resource provisioned in clouds under predefined QoS constraints. Specifically, the objective is two-fold. The first step of resource allocation is resource provisioning which is responsible for determining the number of resource reserved in clouds. We seek model-based analytical prediction methods to guide the resource reservation

in clouds with QoS constraints in the resource provisioning phase. The second step is resource scheduling that makes decision on which resource should be used for each user request (e.g., mapping VMs into PMs in IaaS clouds or scheduling video chunks into CPU cores in video transcoding in clouds). In the resource scheduling phase, this thesis uses heuristic adjustments to compensate the mismatch between the model-based resource reservation and the practical QoS in the system runtime.

1.3 Thesis Contributions

Based on above discussions, this thesis studies three topics of resource allocation in clouds. The first topic is a resource allocation method where memory is the dominant resource in resource allocation in order to guarantee the performance of memory-intensive applications and reduce the total resource provisioned. We formulate the problem of resource provisioning for big memory clouds with consideration of the unique performance behavior of memory-intensive applications. Then, we develop a Markov Chain model to study the resource allocation in big memory clouds and optimize the resource usage by performing VM migration and resource overcommit. Finally, we design an online algorithm, BigMem, to implement our proposed approach. Our work represents the first attempt to achieve efficient resource provisioning that supports big memory clouds.

To serve different applications, in the second topic of this thesis, we propose a resource allocation method for heterogeneous workloads in clouds. In this work, jobs with different dominant resource types (such as CPU-intensive and memory-intensive) are considered simultaneously. In order to support heterogeneous VMs with different shares on different resource types, we firstly propose a flexible VM offering scheme and then define a skewness factor as the metric to characterize the degree of resource balance. Secondly, we develop a Markov Chain model to estimate the minimum number of PMs for heterogeneous

workloads within an acceptable VM allocation delay. Lastly, we design a scheduling algorithm to guarantee the QoS in system runtime.

In the third topic, we turn to a more specific resource allocation problem in clouds, which is cloud-based online video transcoding. The main contributions of this work are threefold. Firstly, we develop an analytical prediction model for cloud-based online video transcoding. By solving the optimization problem with QoS constraints, we are able to find the optimal proportions of transcoding modes and predict the number of needed CPU cores. Secondly, based on the prediction, we design a QoS-aware scheduling algorithm to process the video streams with strict QoS guarantee. Thirdly, we implement a system prototype of COVT to validate the effectiveness with real-world data set. Besides, we also perform simulation studies to evaluate COVT under large scale data sets. The experimental and simulative results show that our proposed system effectively provisions suitable number of resources under predefined QoS constraints, which allows video service providers to offer high-quality services with minimum costs.

1.4 Thesis Organization

The rest of the thesis is organized as follows:

- **Chapter 2** gives a literature review on the related topics.
- **Chapter 3** introduces the work of resource allocation for memory-intensive applications in clouds.
- **Chapter 4** presents the study of resource allocation for heterogeneous workloads with a constraint of resource allocation delay.
- **Chapter 5** discusses the dynamic resource reservation scheme for online video transcoding services in clouds with strict delay requirements.

- **Chapter 6** concludes the thesis and discusses possible future research directions.

Chapter 2

Literature Review

2.1 Homogeneous Resource Allocation

In the field of resource allocation in IaaS clouds, the main research problem can be divided into two parts: resource scheduling and resource provisioning. Resource scheduling is responsible for mapping VMs into PMs under specific goals. Resource provisioning is used to determine the required resource amounts for cloud workloads.

In resource scheduling methods, bin-packing is a typical VM scheduling and placement method that has been explored by many heuristic policies [2, 3, 4, 5] such as first fit, best fit and worst fit, and others. Some recent studies [3, 6] shown that the impact on resource usage among various heuristic policies is similar. The advantage of these basic bin-packing methods for resource scheduling is very simple and stable. The problem is that there is not any consideration of QoS or efficiency in terms of utilization of provisioned resources.

Some recent works investigated scheduling of jobs with specific deadlines [7, 8, 9, 10]. As cloud workload is highly dynamic, elastic VM provisioning is difficult due to load burstiness. Ali-Eldin et al. [11] proposed using an adaptive elasticity control to react the sudden workload changes. Niu et al. [12] designed an elastic approach to dynamically

resize the virtual clusters for HPC applications. These methods are shown to be effective on specific performance objectives. However, none of these scheduling methods is able to consistently offer the best performance for all workload patterns. Thus, Deng et al. [13] recently proposed a portfolio scheduling framework that attempts to select the optimal scheduling approach for different workload patterns with limited time. However, these policies cannot apply directly to heterogeneous resource provisioning because they may cause resource usage imbalance among different resource types.

Another part of resource allocation is resource provisioning which aims at providing appropriate number of resource for certain workloads. To achieve green and power-proportional computing [14], cloud providers always seek elastic management on their physical resources [15, 16, 17, 18, 19]. Li et al. [15] and Xiao et al. [18] both designed similar elastic PM provisioning strategy based on predicted workloads. They adjust the number of PMs by consolidating VMs in over-provisioned cases and powering on extra PMs in under-provisioned cases. Such heuristic adjusting is simple to implement, but the prediction accuracy is low. Model-based PM provisioning approaches [20, 16, 21, 17], on the other hand, are able to achieve more precise prediction. Lin et al. [16] and Chen et al. [21] both proposed algorithms that minimize the cost of data center to seek power-proportional PM provisioning. Hacker et al. [20] proposed a hybrid provisioning for both HPC and cloud workloads to cover their features in resource allocation (HPC jobs are all queued by the scheduling system, but jobs in public clouds use all-or-nothing policy). However, these approaches only consider CPU as the dominant resource in single-dimensional resource allocation.

In the first work in this thesis, we optimize the resource usage in clouds with fully consideration of the features of memory-intensive applications. We reduce the total required resources in clouds by VM migration and resource overcommit. Meanwhile, the

SLA parameters, resource allocation delay and performance degradation, also have been complied with.

2.2 Heterogeneous Resource Allocation

After we proposed a resource management method for memory-intensive applications in clouds. A problem naturally came out, which is the joint resource management of different workloads in clouds. Since CPU- and memory-intensive application have different dominant resource, homogeneous resource management will waste the non-dominant resource significantly. This motivate us to study the resource management for heterogeneous workloads.

There have been a number of attempts made on heterogeneous resource allocation [22, 23, 24, 25, 26, 27] for cloud data centers. Dominant resource fairness (DRF) [24] is a typical method based on max-min fairness scheme. It focuses on sharing the cloud resources fairly among several users with heterogeneous resource requirements on different resources. Each user takes the same share on its dominant resource so that the performance of each user is nearly fair because the performance relies on the dominant resource significantly. Motivated by this work, a number of extensions based on DRF have been proposed [23, 27, 26]. Bhattacharya et al. [27] proposed a hierarchical version of DRF that allocates resources fairly among users with hierarchical organizations such as different departments in a school or company. Wang et al. [23] extended DRF from one single PM to multiple heterogeneous PMs and guarantee that no user can acquire more resource without decreasing that of others. Joe et al. [26] claimed that DRF is inefficient and proposed a multi-resource allocating framework which consists of two fairness functions: DRF and GFJ (Generalized Fairness on Jobs). Conditions of efficiency for these two functions are derived in their work. Ghodsi et al. [25] studied a constrained max-min

fairness scheme that has two important properties compared with current multi-resource schedulers including DRF: incentivizing the pooling of shared resources and robustness on users' constraints. These DRF-based approaches mainly focus on performance fairness among users in private clouds. They do not address the skewed resource utilization.

Zhang et al. [28, 29] recently proposed a heterogeneity-aware capacity provisioning approach which considers both workload heterogeneity and hardware heterogeneity in IaaS public clouds. They divided user requests into different classes (such as VMs) and fit these classes into different PMs using dynamic programming. Garg et al. [30] proposed an admission control and scheduling mechanism to reduce costs in clouds and guarantee the performance of user's jobs with heterogeneous resource demands. These works made contributions on serving heterogeneous workloads in clouds. But they did not consider the resource starvation problem which is the key issue in heterogeneous resource provisioning in clouds.

Thus, in the second work in this thesis, we propose a novel approach (SAMR) to allocate resources with a skewness-avoidance mechanism to further reduce the PMs provisioned for heterogeneous workloads with acceptable resource allocation delay.

2.3 Cloud-based Video Transcoding

Since the first two works studies general workloads in clouds, how to manage resource for specific applications is still a problem that need to be addressed because the relationship between performance and cost of different applications are different. Thus, in the third topic in this thesis, we study a resource allocation method for cloud-based video transcoding systems.

A number of attempts have been made on the problem of video transcoding in clouds [31, 32, 33, 34, 35, 36, 37, 38, 39, 40]. Garcia et al. [36, 37] and Kim et al. [31] proposed to

use MapReduce parallel computing tools (e.g., Hadoop) to speed up the video transcoding process. These works made effort to enhance the performance of off-line video transcoding based on MapReduce or Hadoop. MapReduce is a general framework that splits a big task, dispatches sub-tasks into multiple VMs and collects the results back. It has no support for QoS and it does not dynamically adjust computing resource. Since it is designed for general purpose, the overhead would be large and the QoS support for online video transcoding would be weak. MapReduce based methods are more suitable for offline video transcoding, where all the data are stored in local storages, but they are not appropriate for our considered QoS-aware online video transcoding scenario.

Pereira et al. [35] designed an architecture for processing videos in clouds including merge&split operations. Similarly, Zhuang et al. [34] also designed an architecture for video transcoding in content delivery networks. Ashraf et al. [32, 33] studied the admission control problem for video streams to prevent blockage of services and the cost-efficient resource provisioning problem for transcoding tasks in clouds. Jokhio et al. [39] studied the basic dynamic allocation and release of VMs and the decision making on whether performing transcoding tasks in advance so as to avoid excess storage on cloud servers. All these researches on cloud-based video transcoding are devoted to deal with off-line video transcoding jobs in clouds but do not consider QoS in online video transcoding.

To the best of our knowledge, very few research has studied the problem of provisioning the minimum resources while satisfying restrict QoS for cloud-based online video transcoding. One most related work is [41] where Zhang et al. designed an energy-efficient job dispatching algorithm in transcoding-as-a-service cloud. The video transcoding is viewed as services provided by transcoding engines in the clouds. As each video transcoding task consumes a portion of energy in cloud servers, the authors try to minimize the total energy consumption by intelligently dispatching transcoding jobs to service engines. Meanwhile, maintaining low delay is also a significant constraint factor. However, there

is no real experimental evaluation of the proposed method. Since some assumptions like that the energy consumption of data center is determined by CPU speed might not be completely correct, the practical energy consumption may differ as predicted. Besides, the impact of workload dynamicity is not considered, which also has high chance to break the QoS for online transcoding.

Thus, in the third work in this thesis, we design a system, COVT, that fully considers the feature of cloud environment by adopting infrastructure-aware performance profiling and dynamic task scheduling to guarantee the QoS. Performance profiling methodology is common in evaluating the cloud computing performance on general workloads [6, 42, 43, 44]. Nevertheless, the performance requirements in an online video transcoding system are significantly different from the previous works that mainly target at Mapreduce framework, lacking of QoS guarantee. Thus, the performance profiling method in our system COVT fully considers the unique system configurations of video transcoding tasks.

Chapter 3

Efficient Resource Management for Memory-Intensive Applications in Clouds

In this chapter, we introduce the details about our proposed resource provisioning method for big data clouds, BigMem, in the following aspects: Section 3.1 gives backgrounds and motivations of this chapter. Section 3.2 introduces the features of big data clouds and the impact of memory resource on the performance with a illustrating example Memcached. Section 3.3 gives an overview of BigMem. Section 3.4 derives the model for provisioning resource considering the overhead of VM migration and resource overcommit. Section 3.5 evaluates the performance of BigMem by simulations with different workload patterns. Section 3.6 concludes this chapter.

3.1 Introduction

Recently, as the explosive growth of all kinds of data that generated from billions of personal computers, enterprize servers, mobile devices and sensors, we have witnessed various big data processing applications such as large-graph processing in social networks [45], data analysis [46, 47], high-volume video processing [48] and biomedical informa-

tion processing [49]. The problem of how to economically and fast process big data has attracted much attention from academia and industry (e.g., Facebook, Google, Twitter and IBM, etc.). As a consensus, cloud computing [50] holds great promise to be the big data processing technology because of elastic resource provisioning and economical maintenance. Besides, to achieve high-throughput and low-delay processing of big data applications, in-memory processing [51, 52, 53, 54, 55] is proposed to host big data in the main memory of cloud servers. We refer to such memory intensive applications in clouds as the *big memory clouds*.

Due to large volume of data set in big memory applications, it requires to lease large amount of resources in terms of VMs in clouds. Thus, how to manage resources in clouds for such applications is a key problem that impacts on both monetary costs and performance. On one hand, provisioning too many resources will cause unnecessary energy consumption in clouds as well as costs of users. On the other hand, provisioning too few resources causes poor performance. For instance, if a computing job is allocated with less resource than it required, the performance would be much worse or the job even crashes.

However, current resource management in clouds [19, 18, 16, 56, 20, 57, 11] or big data clouds [47, 46, 58, 59, 60, 61, 62] are not suitable for supporting memory-intensive applications. This is due to the fact that the resource management for memory-intensive applications has its unique performance requirements compared with traditional applications. These unique features of memory provisioning is experimentally illustrated and detailed discussed in Section 3.2. Existing resource management methods based on other resource types are not possible to be applied for big memory clouds. Thus, the resource provisioning for big memory clouds is necessary to consider memory as the first-class resource to ensure well performance. Besides, current attempts [47, 46, 58, 59, 60, 61, 62, 63, 59]

on resource management for big memory clouds still did not provide a data center-wide solution to optimize the resource usages.

Motivated by above analysis, we propose a resource-conserving resource management approach, namely *BigMem*, for big memory clouds in this chapter. BigMem is a IaaS cloud resource management scheme that estimates and optimizes the minimum number of active PMs required for VM requests by memory-intensive applications. BigMem uses a basic Markov Chain model with two extensions, resource overcommit and VM migrations, to analytically study the resource usage in cloud data center. To guarantee the memory-intensive applications' performance, we define two SLA metrics in BigMem as the constraints in optimization: VM allocation delay and performance degradation. By solving the model with the preset SLA constraints, minimum number of active PMs is obtained. We evaluate our solution with both synthetic and real-world workloads. The results show that BigMem is able to effectively provision less resources while satisfying the SLA requirements. On average, BigMem reduces the resource usage by approximately 63% and 27% compared with the peak-load provisioning and Auto-scaling approach, respectively.

3.2 Big Memory Clouds

Main memory has been one of the most critical resource components for various systems and applications. With the recent popularity of cloud computing, researchers have started to pay more attention to develop cloud-based memory-intensive applications. For example, social networks [45], web caches [64], data analysis [46, 47], large-volume video processing [48] and biomedical information processing [49] are typical ones. They generally require a large amount of memory to execute and CPU is considered to be redundant in such applications. Besides, the trend that the computing capability advances faster than

Figure 3.1: Experiments results of memcached.

memory capacity has continued for years. The gap accumulated over the years has made memory resources becoming the bottleneck for many data-intensive applications [51].

To illustrate the different performance behaviors of memory management compared with CPU, we performed experiments on a data cache system named *memcached* [65] as a motivating example. The experiments were conducted on a cluster of 8 nodes with 10 Gbps inter-node network bandwidth. Each node has a six-core Xeon E5-1650 CPU and 16GB DRAM. The workload contains *get* and *set* operations uniformly distributed on the whole working set. Based on the results given in Fig. 3.1, we make the following key observations.

- Firstly, the main memory capacity is the key factor for the performance in memory-intensive applications. In Fig. 3.1.a, we allocate different size (from 1 to 16 GB) of memory to the data cache system with 8 GB data set in one single node. A big memory cloud system usually requires sufficient RAM space to host data. If there is not sufficient memory for the data cache system, the throughput of data accesses may degrade significantly because of the overhead of data swap between disk and

main memory. Thus, considering to satisfy memory demands for such applications is most crucial in resource provisioning.

- Secondly, the working set hosted in multiple PMs shows performance degradation for some big memory applications [58]. This is different from CPU cores allocation that can cross multiple PMs with minimum impact on the performance [20]. In Fig. 3.1.b, the throughput degrades significantly as the size of cluster increases from one to multiple nodes. The reason for performance degradation is mostly due to the excessive network delay caused by distributed data locations.
- Thirdly, the impact of overcommit is high. Overcommit has been considered as an effective way to support more applications with limited memory resources. It takes the advantage that not all applications utilizes the requested memory at all time, and thus additional applications can be admitted to utilize the available memory with the hope that the total requested amount does not exceed the physical limit [66, 67]. While overcommit offers more effective use of memory resources, it risks performance degradation when the total requested amount exceeds the physical limit (e.i., overload). When that happens, remote memory resources will be sought resulting excessive delay in memory access. Fig. 3.1.c shows the mean performance degradation against different overcommit factors of memcached with 8GB working set. The overcommit factor is defined as the ratio of overcommitted resource to the required resource of the application [66]. This phenomenon implies that though overcommit is cost-efficient for big memory clouds, the risk of overload [67, 68] should be fully taken into accounts.
- Fourthly, the overhead of VM migration is directly determined by the size of memory image in the VM [69]. While VM migration is commonly used to consolidate the resource usage [70] in data center to reduce the power consumption, the memory

size of VM should be a key indicator in designing migration algorithm. As frequent resource allocation and deallocation, there would be small holes of idle resources in PMs after a long run. In this chapter, we use VM migration to eliminate the memory holes caused in data center in the runtime in order to conserve resources.

This chapter focuses on resource management for supporting above unique performance behaviors in big memory cloud systems. As the memory resource is the first-class consideration, the users performance are well guaranteed while the costs of cloud operator are also reduced with the memory-based resource management approach BigMem.

3.3 System Overview

In this section, we provide an overview of the proposed algorithm BigMem. Table 3.1 lists the key notations used throughout this chapter.

We consider the scenario where users develop and deploy their memory-intensive applications in clouds by reserving VMs in a pay-as-you-go manner according to memory consumption (assume that CPU is efficient). Users can acquire or release a VM in an on-demand manner and pay according to the VM types with different RAM sizes (e.g., Rackspace [71]). The total number of PMs in the data center is N , each of which has M (GB) of RAM. The workloads consist of tons of user requests for different types of VMs. A VM request is accepted if the allocator successfully finds enough resource in the active PM list. Otherwise, the request will be delayed until additional PMs are switched on. We refer to the delayed requests as overflowed requests in this chapter. The fewer PMs provisioned, the more overflowed requests and longer resource allocation delay will occur. Thus, there is a trade off between the number of PMs and resource allocation delay.

Ideally, cloud providers should provide adequate number of PMs so that user requests can be immediately accommodated. However, due to the fluctuation in workload, it

Table 3.1: Notations of BigMem algorithm

N	Number of PMs in the considered data center
K	Number of VM types
M	Memory capacity in GB of a PM
r	Current total available size of memory in a PM
$\{r\}$	A state with r available memory in Markov Chain
$p(r)$	The steady probability of state $\{r\}$
b_i	Memory size of type- i request and VM in GB
v_i	Number of type- i requests
λ_i	Arrival rate of type- i requests
μ_i	Service rate of type- i requests
O_{mgr}	Migration overhead in a time slot
O_f	Over-commit factor
$T_{ij}(t)$	j^{th} continuous dynamic memory usage for type- i request, t is time
$D_{ij}(e)$	j^{th} discrete dynamic memory usage for type- i request, e is time epoch
$P_D(x)$	Total performance degradation in x^{th} PM
$P_{D_{mgr}}$	Average performance degradation caused by migration for each PM
$P_{D_o}(x)$	Performance degradation caused by overcommit in x^{th} PM
P_{D_o}	Average performance degradation caused by overcommit
P_O	Overflow probability
n	provisioned number of active PMs ($n \leq N$)
α	The predefined P_O threshold
β	The predefined P_D threshold

is impossible to always guarantee immediately accommodation unless significant over-provisioning of PMs is involved. Thus, overflowed requests will suffer long resource allocation delay, which is negative for big memory cloud users' experience. Besides, overcommit and VM migration also affect the experience of users. In this chapter, we define two SLA to users, VM allocation delay and performance degradation, that should be satisfied by the resource scheduling and provisioning. Both delay and degradation are measured by time.

Then, the research problem of this chapter is to provision minimum number of PMs n for the workloads under the condition that two SLA metrics are satisfied. This optimization is on the perspective of cloud providers such as Rackspace who benefits from

economical resource provisioning scheme by BigMem. On the view from users, two SLAs are satisfied to ensure the performance of their applications.

The flowchart of the BigMem algorithm is illustrated in Fig. 3.2. We first seek model-based approach to estimate the number of active PMs with predicted workload. We recognize the potential variation between the predicted and actual workloads that may cause under or over provisioning of active PMs. To minimize this impact, overflowed requests must be treated promptly. The compensator may immediately power on an adequate number of PMs when overflowed requests occur. The function of each component is summarized as follows.

Figure 3.2: Flowchart of BigMem algorithm.

Workloads. The workloads consist of many requests for different types of VMs. The cloud provider offers a range of VM types with different memory capacities and

with different charging rates. We assume that the cloud provider offers K VM types. For each VM type, the memory capacity is provisioned with b_i GB ($i = 1, 2, \dots, K$). In brief, we represent the VM offering using a vector $\vec{b}$, with the memory capacity in an ascending order ($b_i \leq b_{i+1}$, $1 \leq i < K$). As the pay-as-you-go nature, we model an request submitted by a user as a type- i request for a VM with b_i GB of memory.

Workload predictor. For convenience, we divide the operating time of the cloud into equal length time slots. The resource provisioning is conducted at every time slot. The workload predictor predicts the workloads amounts for each type of requests in the coming time slot based on the history data. In the literature, there are many available methods proposed for load prediction [18, 72]. In BigMem, we pick prediction algorithm Exponential Weighted Moving Average (EWMA) as the workload prediction method.

EWMA is a common method used to predict an outcome based on history values. At a given time z , the predicted value of a variable can be calculated by

$$E(z) = w \cdot O_b(z) + (1 - w) \cdot E(z - 1), \quad (\text{Eq. 3.1})$$

where $E(z)$ is the prediction value, $O_b(z)$ is the observed value at time z , $E(z - 1)$ is the previous predicted value, and w is the weight. Thus, we can get the arrivals for each type of VMs in the coming time slot based on history data.

Algorithm 1 Provisioner algorithm of BigMem

```

1: if The current time is beginning of a time slot then
2:   Predict the workload;
3:   for  $n = 1$  to  $N$  do
4:     Compute  $P_O$  with model in Section 3.4;
5:     if  $P_O \leq \alpha$  then
6:       Provision  $n$  in the coming time slot;
7:       Break;

```

PM provisioner. At the beginning of each time slot, BigMem estimates the rough number of active PMs in Algorithm 1. One of the SLA, resource allocation delay, should

be satisfied in provisioning phase. The VM allocation delay may be affected by many factors such as scheduling algorithm, VM initialization and queuing delay, etc. In this chapter, we mainly focus on the queuing delay caused by under-provisioned PMs. Due to the workload burst, overflowed requests are inevitable. In resource estimation model of BigMem, we define overflow probability (P_O) to be the probability that a VM request is unable to schedule immediately due to lack of vacancy in the active PMs. Since P_O and delay are convertible, we use P_O to present delay in the model. To reduce the chances that a user experiences delayed service, we should maintain the condition that $P_O \leq \alpha$ with α being set adequately low. We can get the minimum n that satisfy the condition by running the model introduced in Section 3.4 with different n . The details of the provisioner are given in Section 3.4.

Algorithm 2 Allocation algorithm of BigMem

```

1: if Each request for type- $i$  VM then
2:   Compute  $P_{D_{mgr}}$ .
3:   for  $x = 1$  to  $n$  do
4:     Compute  $P_{D_o}(x)$ 
5:      $P_D(x) = P_{D_o}(x) + P_{D_{mgr}}$ 
6:     if  $P_D(x) \leq \beta$  then
7:       allocate  $b_i$  in  $x^{th}$  PM;
8:   if No PM can host the request then
9:     if  $\sum_{j=1}^n r_j \geq b_i$  then
10:      find the  $x^{th}$  PM that with maximum  $r$ 
11:      Migrate  $(b_i - r_x)$  GB memory from  $x^{th}$  PM to other machine,  $r = r + b_i - r_x$ ;
12:      Allocate  $b_i$  GB memory in  $x^{th}$  machine,  $r = r - b_i$ ;
13:   else
14:     Delay the request;
15: if A type- $i$  VM completes execution then
16:   Release the memory occupied by the VM,  $r = r + b_i$ ;

```

Job scheduler. Job scheduler in BigMem is a first fit (FF) VM scheduler which maintains a list of all available (active) PMs and searches for RAM vacancy in the list sequentially for each user request. If there is available resource, the VM request will be

hosted in corresponding PM. If no PM can satisfy the RAM demand of the request, it becomes an overflowed request. We consider to use VM migration and resource overcommit to further reduce the number of required PMs. The migration and overcommit both cause performance degradation to the VMs. Thus, the other SLA, performance degradation P_D , is satisfied in scheduling phase. P_D is defined as the ratio of additional execution time to the total execution time of a VM. Similar to delay constraint, the optimization operations can impose a condition that $P_D \leq \beta$ where β is a set threshold. The detailed scheduling process is listed in Algorithm 2 and described as follows: 1) For each type- i request, the practical memory usage demand $D_{ij}(e)$ is a continuous curve that we estimate from the past based on the existing prediction algorithm, Exponential Weighted Moving Average. Given the workload amounts, BigMem processes the resource demand curve by discretizing the curve into bars with equal-length epochs. The value of the bar is computed by the mean value of the curve in each epoch. After the discretization, the memory demand of each VM request is represented as a vector of memory usage at different epochs. 2) The dynamic resource usage distribution allows us to overcommit the resource and serves more VMs in a PM. The total number of VMs in a PM is limited by $P_D \leq \beta$ which prevents high performance degradation. This mechanism will finally find an optimal cost scheme while meeting the QoS. 3) The migration operations are triggered in the cases that there is no available memory in any single PM to host a request, and the overall free memory size in the provisioned data center is sufficient to host the VM. Therefore, migration avoids powering up extra PMs at the cost of some migration overhead. If the request cannot be hosted in a single PM, BigMem will check whether the request can be served after migrations. VM migration in BigMem follows a greedy approach that always finds the PMs with most available memory in provisioned PM list. 4) If a request cannot be accepted by consolidating VMs in the current machine list, this request is overflowed and delay in service is resulted. 5) When a VM is released, all the

resources that the VM occupied will be released. The released resources can be reused for other requests.

Compensator. After the provisioning prediction is produced by the provisioner, the cloud system starts allocating the real workloads. When all active PMs are nearly full and cannot serve additional jobs, the additional jobs would be overflowed into a queue waiting for extra PMs to be powered up. In such a case, delay in services will be resulted. While cloud providers may specify an SLA to permit a certain percentage of requests to experience delay in services, the bursting workload may bring a short term high number of requests causing an excessive request overflow. To ensure that the committed SLA can be met even under the unknown behavior of workload, a heuristic-based adjustment can be employed to preemptively increase the number of active PMs before overflowed requests occur.

3.4 System Modeling

In this section, we present the analytical model to determine the SLA value P_O given a particular number of active PMs in PM provisioner. Unlike other works that develop models for computing resource management [20, 73, 74, 16, 21], our model focuses on memory resource, with special consideration on two unique features of big memory cloud systems (migration and overcommit). We firstly design a Markov chain model as a base model to describe BigMem with a basic FF algorithm, and then extend the base model to further capture migration and overcommit.

3.4.1 The Base model

In our base model, we focus on FF without virtual machine migration or overcommit. We consider a data center with N PMs, each of which has M GB RAM. Without loss

of generality, we assume $M \geq b_K$ (a PM can host the VM with the largest memory resource).

Similar to the previous works [20, 73, 74], our analytical model assumes that the arrivals of type- i requests follow a Poisson process with a rate of λ_i ($i = 1, 2, \dots, K$). The service time of a type- i request follows an exponential distribution with a rate parameter of μ_i ($i = 1, 2, \dots, K$). That means, the lifetime for a type- i VM follows an exponential distribution with a rate parameter of μ_i ($i = 1, 2, \dots, K$).

Considering the fluctuation of the resource requirements, it is challenging to study the resource utilization of all PMs in data center as well as estimating the allocation delay. Given a data center with N PMs and each with M GB of RAM, modeling all PMs in such a system results a system state space of order $O((M/b_1)^N)$ which is mathematically intractable. We observe that in FF algorithm, each new type- i request arrival searches the active PM list sequentially to find a match on the resource requirement for b_i . If a PM can accommodate the arrival, the request is admitted. Otherwise, the next PM in the list is considered, and this search process continues until the request reaches the last PM in the list. If the request remains unaccommodated by the last PM, the request is overflowed. With this observation, it permits a continuous time Markov chain model focusing solely on a particular PM where its arrival is the overflow leaking from its previous PM. The first PM in the list requires a different consideration where its arrival is simply the overall arrival from all users. We illustrate this modeling approach in Fig. 3.3. The arrival of a particular PM except the first is based on the overflow from its previous PM. Since the overflow requests from the last PM cannot be served, these requests are overflowed. These overflowed requests form the overflow probability which is P_O .

We use a one-dimensional state space to describe the evolution of memory usage for a particular PM. The state $\{r\}$ represents the amount of memory available in a PM, where

Figure 3.3: The Base Model for BigMem with a FF scheduling policy.

$r \in 0, 1, 2, \dots, M$. Given a particular state $\{r\}$, the total amount of memory occupied in the PM is thus $M - r$. Since a PM may be occupied by several VMs, we denote the expected number of type- i VMs in a PM by v_i . Each memory allocation/release operation triggers a system state transition. In the following, we describe the memory operations and the corresponding system state transitions in a PM. We begin by defining an indicator function $I(x)$ in the following for our subsequent formulation, where

$$I(x) = \begin{cases} 1, & x \geq 0 \\ 0, & \text{otherwise.} \end{cases} \quad (\text{Eq. 3.2})$$

The evolution of the system state is governed by request arrivals and departures. We first denote $R\{s|r\}$ to be the rate of transition from state $\{r\}$ to state $\{s\}$. Upon an arrival of a type- i request, the request is admitted if there is an available memory block in the PM meeting the requirement, that is, $b_i \leq r$. The transition occurs in this case from state $\{r\}$ to state $\{r - b_i\}$. The transition rate is given by $R\{r - b_i|r\}$, where $i = 1, 2, \dots, K$ and

$$R\{r - b_i|r\} = \lambda_i \cdot I(r - b_i). \quad (\text{Eq. 3.3})$$

The release of memory occurs when a VM terminates. The rate of memory release depends on the number of VMs currently active in a PM. At a particular state $\{r\}$ where

$r \leq M$, we know that there is $M - r$ amount of memory utilized. Based on our model, the number of a particular type in service is proportionate to its utilization of the system. Thus the expected number of type- i VMs in service in a PM can be computed by

$$v_i = \frac{\frac{\lambda_i}{\mu_i}}{\sum_{i=1}^K \frac{\lambda_i(M-r_i)}{\mu_i}} \cdot (M - r) \quad (\text{Eq. 3.4})$$

with an overall departure rate of $v_i \mu_i$ for type- i VMs.

Upon a departure of a type- i request, the system state transits from state $\{r\}$ to state $\{r + b_i\}$. Thus, the possible transitions triggered by VM departures are

$$R\{r + b_i|r\} = v_i \mu_i \cdot I(r + b_i) \quad (\text{Eq. 3.5})$$

where $i = 1, 2, \dots, K$.

The above expressions permit construction of a $(M/b_1 + 1)$ -by- $(M/b_1 + 1)$ infinitesimal generator matrix (Q) for the CTMC model. The steady-state probability of each state, $p(r)$, can be solved numerically according to the following balance equation set.

$$\begin{aligned} p(r) \cdot \left[\sum_{i=1}^k (v_i \cdot \mu_i \cdot I(r + b_i) + \lambda_i \cdot I(r - b_i)) \right] = \\ \sum_{i=1}^k [p(r + b_i) \cdot \lambda_i \cdot I(r + b_i) + p(r - b_i) \cdot v_i \cdot \mu_i \cdot I(r - b_i)]. \end{aligned} \quad (\text{Eq. 3.6})$$

Solving the steady probabilities of the system allows us to study the high-level performance of the system. The memory utilization of a PM can be determined by

$$U = \sum_{i=0}^M p(i) \cdot (M - i). \quad (\text{Eq. 3.7})$$

Let P_{O_i} be the overflow probability of type- i requests, given below

$$P_{O_i} = \sum_{r=0}^M p(r) I(b_i - r). \quad (\text{Eq. 3.8})$$

The overall overflow probability for all types, P_O , is

$$P_O = \frac{\sum_{i=1}^k P_{O_i} \lambda_i}{\sum_{i=1}^k \lambda_i}. \quad (\text{Eq. 3.9})$$

In the following, we shall extend the base model to capture migration and overcommit.

3.4.2 Migration Overhead

In the case where an arriving request with type- i is larger than the total available memory of the PM r ($b_i > r$), the request may still be admitted if other resident VMs in the PM can be migrated to another PM. We shall make certain adjustments in our base model to capture the VM migration operation.

Upon admitting a new request of type- i with migration involvement, the system transits from a state $\{r\}$ to a state $\{0\}$ indicating that some VMs are forced to migrate in order to make just enough room for the new request to be admitted. Additionally, this operation triggers migration of $b_i - r$ amount of memory on average to another PM. Specifically, we can view the entire cluster with n machines as a memory pool. Thus, the base model is used to study the resource pool with $(n \cdot M)$ RAM resource to calculate the overflow probability. Based on the solution given earlier for the base model, we estimate the total migration amount in GB, O_{mgr} , by

$$\begin{aligned} O_{mgr} = & \sum_{x=1}^n \sum_{i=1}^{b_k-1} \cdot p(x, i) \cdot \left(\sum_{j=2}^k (b_j - i) \cdot \lambda_j \cdot I(b_j - i) \right) \\ & \cdot \sum_{y_1=1}^i \dots \sum_{y_{x-1}=1}^i \sum_{y_{x+1}=1}^i \dots \sum_{y_n=1}^i G(p(1, y_1), \dots, \\ & p(x-1, y_{x-1}), p(x+1, y_{x+1}), \dots, p(n, y_n)) \end{aligned} \quad (\text{Eq. 3.10})$$

where $p(x, y)$ denotes the steady probability of state $\{y\}$ for the x^{th} machine in the server list. The function $G(\cdot)$ is given in Eq. 3.11, which computes the probability that a migration occurs in system. The summations in Eq. 3.10 enumerate all the possible time period combinations of the VMs in a PM. These summations have a high computational cost. Fortunately, they can be computed off-line because they are workload independent.

Thus, the overhead of these summations is eliminated from the runtime overhead.

$$\begin{aligned}
 & G(p(1, y_1), \dots, p(x-1, y_{x-1}), p(x+1, y_{x+1}), \dots, p(n, y_n)) \\
 &= \begin{cases} \prod_{l=1, l \neq x}^n p(l, y_l), & \sum_{l=1, l \neq x}^n p_{l, y_l} \geq (b_j - i) \\ 0, & \text{otherwise.} \end{cases}
 \end{aligned} \tag{Eq. 3.11}$$

Then the performance degradation caused by migration overhead is

$$P_{D_{mgr}} = \frac{O_{mgr}}{(\sum_{i=1}^K \mu_i / K) \cdot B} \tag{Eq. 3.12}$$

where B is the average inter-machine bandwidth of network in the data center. It is straightforward to adapt the calculation to the migration with different inter-machine bandwidths.

3.4.3 Overcommit

This subsection considers the case for the overcommit of memory resource. Since memory computation pattern estimation is not the main concern of this chapter, we assume the knowledge of memory usage patterns by using existing techniques for load prediction or profiling techniques (e.g. [66, 6, 72]).

In the base model, a memory request is one element b_i from vector $\vec{b}$. We denote the memory usage after discretizing $T_{ij}(t)$ to be $D_{ij}(e), e = 1, 2, 3, \dots, E$, where E is the total number of epochs. We further convert $D_{ij}(e)$ into a probability density function (pdf) $f_{ij}(m), m \in \text{range}(D_{ij})$. This means the constant RAM demand b_i is divided into different values with different probabilities. Let $\vec{b}_{oi}$ and $\vec{\lambda}_{oi}$ be the memory demand vector and arrival rate vectors for type- i request with overcommit. Then we use $f_{ij}(m)$ and the VM block size b_i to generate the input block size vector for type- i request by

$$\vec{b}_{oi} = b_i \cdot f_{ij}(m). \tag{Eq. 3.13}$$

The according arrival rate vector is

$$\vec{\lambda}_{o_i} = \lambda_i \cdot f_{ij}(m). \quad (\text{Eq. 3.14})$$

By using the above adjustments, our base model can be reused. Precisely, in the base model, we obtain the state transitions with the adjusted arrival rate and block size vectors for overcommit and the same computational approach is used to solve the balance equation and the overflow probability.

Since memory overcommit incurs overhead, we shall now compute the performance degradation due to overcommit. With overcommit, the practical total usage in a PM may exceed the capacity of a PM M . That is, the PM may be overloaded. We denote O_{f_x} as the overcommit factor for x^{th} machine that defines the ratio of overloaded time to total execution time. Let C be the number of values in $f_{ij}(m)$ and V be the number of VMs in a PM, $V = \sum_{i=1}^K v_i$. The VM types in x^{th} PM are $t_{x_z}, z = 1, 2, \dots, V$. Then, the overcommit factor in x^{th} PM can be derived as following

$$O_{f_x} = \sum_{c_1=1}^C \sum_{c_2=1}^C \dots \sum_{c_V=1}^C \left[\prod_{w=1}^V \cdot f_{t_{x_w}j}(c_i) \cdot I\left(\sum_{u=1}^K b_{o_i}(c_u) - M\right) \right]. \quad (\text{Eq. 3.15})$$

Eq. 3.15 can also be computed off-line with little runtime overhead. We note that the relationship between overcommit factor and performance degradation is application dependent. In practice, we can use profiling or pre-execution to obtain this relationship (e.g., in Fig. 3.1.c). Given that relationship, the performance degradation caused by overcommit in x^{th} machine $P_{D_o}(x)$ can be obtained.

Since the performance degradation is described by the ratio of delayed time to total execution time, the total performance degradation P_D for a PM can be computed by summing up the overhead caused by migration and overcommit.

$$P_D = \sum_{x=1}^n P_{D_o(x)} + P_{D_{mgr}}. \quad (\text{Eq. 3.16})$$

Since performance degradation postpones the execution of VMs, we need to appropriately add the time overhead to the departure rates $\vec{\mu}$ in the base model to adjust the impact caused by performance degradation. Specifically, all the VMs in the system may be affected by such performance degradation and their execution time may be increased because of delay caused by migration and overcommit. Thus, we need an adjustment to our model in order to capture this effect. This can be achieved by adjusting the departure rates of VMs to $\vec{\mu} - \vec{\delta}_\mu$, where $\delta_\mu = P_D \cdot \vec{\mu} / N_{request}$ and $N_{request} = \sum_{i=1}^K \sum_{u=1}^C \lambda_{o_i}(u)$ is the total requests in a time slot. With this adjustment, we seek fixed-point iteration to numerically solve the adjusted model.

3.4.4 Complexity Analysis

As the extremely large state space of data center (for example, $O((M/b_1)^N)$ for space and $O((M/b_1)^{3N})$ for computation in our settings), current methods can not be applied to capture the details of each machine. BigMem is modeled into a Markov Chain that partitions the state space into per machine level for analysis. Thus, the computation complexity is reduced to $O(N \cdot (M/b_1)^3)$ that is linear with the size of data center. The RAM requirement for solving the model is also reduced to $O(N \cdot (M/b_1)^2)$ which is achievable by a typical machine in today's technology. For example, the computation of a 1000-server cluster in our experiment consumes less than 1 MB DRAM, and takes 30 seconds on average for each model computation at every beginning of a time slot. With the reasonable low runtime overhead, BigMem is able to support data centers with a large number of PMs.

3.5 Evaluation

In this section, we evaluate the effectiveness of our approach. Our goal is two-fold: firstly, we confirm the accuracy for BigMem on resource provisioning for dynamic workloads; sec-

ondly, we demonstrate the effectiveness of our approach in terms of resource optimization with a particular SLA for big memory cloud systems.

3.5.1 Methodology

In our experiments, we develop a trace-driven simulator for modeling memory requests and allocations in a data center. Particularly, we consider a data center containing 1000 ($N = 1000$) PMs, each with 16 CPU cores and 128 GB ($M = 128$) RAM. The VM types in our simulation follow the memory configuration of Rackspace (i.e., $\vec{b} = [1, 2, 4, 8, 15, 30]$). There are a number of parameters for sensitivity studies. The default settings are: the duration of a time slot is one hour; P_O and P_D are set to 3% and 5%, respectively. The sensitivity of these settings are separately studied in Section 3.5.3.

Workloads. In order to assess our algorithms, we use six synthetic workloads including four basic patterns (namely stable, growing, pulse and wave) and two patterns re-generated according to real workloads, as shown in Fig. 3.4. This figure shows the average total memory usages in data center against time slot (24 slots). Different types of VM requests in each time slot is distributed as the power of two in descending order in our settings.

For the two real workload traces, we regenerated the memory request patterns from Microsoft workloads from Hotmail and MSR Cambridge. The details of those two workloads have been studied in the previous work [75]. The two real workload traces represent the real workloads from large-scale memory intensive applications with many users. The peak load in the two real workloads have been mapped to six times of the lowest points.

Comparisons. We study the following performance and cost metrics for each time slot: the number of PMs per time slot, mean utilization, overflow probability and performance degradation. We define the mean utilization to be the average utilization of

Figure 3.4: The six workloads with time (hour)

active PMs. To directly assess the effectiveness of one approach on a long-time period, we define *machine hour* to be the total machine time for those active PMs. A large number of machine hours leads to a higher power consumption.

We simulate BigMem and compare the performance of each optimization methods such as VM migration and resource overcommit. Besides, we also run experiments using the real workloads without workload prediction, which is called “BigMem Oracle”. For comparison, the simulation results with workload prediction is presented as “BigMem”. In addition to the above comparison, we also compare the results of BigMem with an heuristic approach that is called **Auto-scaling** [19, 18] in this chapter. Auto-scaling performs the adjustment according to the heuristics based rules. It uses a simple feedback control mechanism to adjust the number of active PMs based on the current states. Two utilization thresholds are used to control the adjustment, namely U_H and U_L . The algorithm periodically inspects the system utilization. When the system utilization exceeds

U_H or falls below U_L , the number of active PMs increases or decreases, respectively, at a particular predefined percentage. Since selection of parameters directly influence the performance, we conduct experiments with a wide range of parameters. We vary U_H in the domain of $[0.8, 0.85, 0.9, 0.95]$ and U_L in the domain of $[0.6, 0.65, 0.7, 0.75, 0.8]$, and the adjustment percentage in the domain of $[1\%, 5\%, 10\%, 15\%, 20\%, 30\%, 40\%, 50\%]$. We consider all combinations of U_H and U_L and present the average, maximum and minimum performance results of the experiments for detailed comparison.

In the following, we present the overall results, followed by the results on sensitivity studies.

3.5.2 Overall Results

In this section, we first study the accuracy of our analytical model. Then, we present the comparison between BigMem and Auto-scaling method. Next, we illustrate the comparison of different optimizations in BigMem, followed by the sensitivity studies.

Table 3.2: Overall results in total machine hours

Workload	BigMem	BigMem Oracle	Peak-load	Auto-scaling (average/minimum/maximum)
Stable	2470	2420	6676	3383/3012/3800
Growing	3530	3490	9590	4832/4306/5440
Pulse	3730	3660	9031	5109/4548/5738
Wave	4160	4120	11043	5688/5048/6413
Hotmail	2640	2540	12335	3616/3216/4051
MSR	6180	6060	12702	8456/7532/9507

Algorithm Validation. In order to validate the accuracy of our developed model for provisioning, we focus on PM provisioner and measure the error which is defined as $error = 1 - (\frac{m'}{m})$, where m and m' are the metric result from real system states (BigMem

Oracle) and BigMem, respectively. A reading of $error = 0$ indicates no difference is found between the simulation and analytical results. On the six workloads, the average errors on all the four metrics of all workloads in our experiments are 1.9%, 0.8%, 2.3% and 1.2% for the number of active PMs, mean utilization, overflow probability and performance degradation, respectively. This result reveals that our model-based provisioner can achieve a high accuracy in provisioning active PMs for different workload patterns.

Overall Results. Table 3.2 shows the overall results on machine hours. For all workloads, the results of our model-based provisioner are very close to that of the simulation results. Thus, our model-based approach is accurate on guiding the provisioning in different workloads. Comparing the results of BigMem with that of auto-scaling algorithm, BigMem reduces the number of active PMs by 18%, 27% and 35% compared to the minimum, average and maximum results from auto-scaling algorithm, respectively. Note that we have excluded the results for auto-scaling algorithm if it violates the SLA. Compared with the peak-load provisioning, BigMem is able to conserve the resources by approximately 63%.

We now study the detailed results in each time slot. Fig. 3.5 illustrates the number of active PMs given by BigMem and Auto-scaling. The vertical lines mark the result range by running all parameter combinations of Auto-scaling. From the figure, we make the following three observations. Firstly, the prediction of BigMem is very close to the optimal results on each time slot. The noticeable difference between BigMem and simulation occurs only in bursting workloads such as the exceptionally high changes in pulse and wave workloads. Secondly, the mean number of active PMs provisioned by Auto-scaling is close to the simulation results *only* when the workload has little variation such as the stable pattern. In the dynamic workload, Auto-scaling either cannot satisfy the SLA or severely over-provisions resources. Although the numbers of active PMs in many configurations of Auto-scaling are lower than those of BigMem and simulations,

Figure 3.5: Number of active PMs in each time slot.

most of them violate the SLA, which can be seen from Fig. 3.6 that shows high overflow probability for Auto-scaling. In contrast, BigMem choose the suitable number of active PMs. These results clearly demonstrate that BigMem can effectively optimize the number of active PMs without excessive violation to the strict SLA. This result holds for both basic workload patterns and real workload patterns.

Fig. 3.6 shows the overflow probabilities of all methods, where these overflow probabilities represent the probability that a request cannot be served immediately and put in the queue in compensator. It is obvious that BigMem can satisfy the P_O SLA requirement of 3% by the provisioner. It also indicates that cloud operators are able to effectively control the delay SLA by setting different thresholds in the provisioner. In contrast, the mean SLA violation of Auto-scaling policies is 43% for all parameter combinations. Rule-based Auto-scaling algorithm fails to meet the SLA when it becomes under-provisioned. For

Figure 3.6: The overflow probability of four synthetic workloads and two real workloads.

instance, when workload increases, most results of Auto-scaling algorithm are unable to react and scale the active PMs up quickly, which leads to significant under-provisioning of active PMs. As a result, P_O increases sharply as the most requests cannot be served. In the descending workload scenarios of pulse and wave workloads, P_O falls below the tolerable level noticeably at the cost of over provisioning.

Individual impacts of optimizations. To assess our provisioning models and its components of resource optimizations by consolidating VMs through migration and overcommit, we consider four variants for of the model: “Basic”, “Mgr-only”, “Oc-only”, and “Both”, which represent our basic model without optimizations, with migration only, with overcommit only and with both optimizations, respectively. The variant “Both” is also referred as BigMem in this section, when appropriate.

Fig. 3.7 shows the overall comparison for the four variants in assessing individual

3.7.a: Number of machines

3.7.b: Utilization

3.7.c: Overflow probability

3.7.d: Performance degradation

Figure 3.7: The results of four components of BigMem for four synthetic workloads.

impacts on performance including overcommit and migration in the model-based provisioner. We focus on the average performance within a time slot. Firstly, the experiments results confirm that both migration (Mgr-only) and overcommit (Oc-only) contribute to cost reduction. With both options included (i.e. Both), the number of active PMs of Both is 34% lower than that of Basic. We also highlight that the effectiveness on reducing the number of active PMs by using the two options together is larger than the sum of using individual option (as can be seen from Fig. 3.7.a).

Secondly, after considering VM migration, a lower number of active PMs can be provisioned. Interestingly, the Mgr-only model can achieve the utilization close to 100%. Utilization (as shown in Fig. 3.7.b) of overcommit only model is a little lower because of

Figure 3.8: Sensitivity studies for workload Stable.

the existence of memory holes. Due to the need for avoiding performance degradation in the overcommit, the system (over and both) attempts to control the P_D caused by memory overcommit within the threshold at the cost of sacrificing utilization.

Thirdly, the SLA requirement can be met by BigMem. P_O reflects the number of jobs being delayed in the system, which is under the threshold in the experiment. The fluctuations in the curve of P_O are sometimes irregular because the value of P_O depends on the amount of workload and the provisioned number of PMs. Thus, the P_O curves fluctuate with the workloads.

3.5.3 Sensitivity Studies

We now perform sensitivity studies on major parameters in BigMem, including α , β and different types of memory requests (small vs. big memory demand) in the workloads. For each experiment, we study the impact of varying one parameter while keeping other parameters to their default settings. We focus on the metric of the number of active PMs. Since we observe similar results on different workloads, we present the results for workload Stable.

Fig. 3.8.a shows the results for varying α , the threshold of P_O . P_O represents the chance of delayed VM allocation which will impact the QoS. This parameter can be

changed in the provisioner in order to adjust the SLA level according to the different requirements on the allocation delay. As expected, the number of active PMs decreases as the overflow probability threshold α grows. This is because a higher threshold permits more requests to experience delay in services, which allows providers to decrease the number of active PMs. It holds elasticity for cloud operators to set an appropriate value of α .

Fig. 3.8.b shows the results for varying β , threshold for P_D . We only show two variants of BigMem (Oc-only and Both). As P_D becomes larger, constraints of migration and overcommit are relaxed, and thus more requests can be hosted within a PM, as a result, the number of active PMs of both approaches decreases. Comparing Oc-only and Both, we can see more migration activities further reduce the number of active PMs.

Fig. 3.8.c shows the results for varying the request types with different amount of small and big requests. The total RAM demand is fixed and we only allow the smallest and largest VM types. To be specific, the number of type-1 requests varies from 0 to 24000 and the number of type-6 requests reduces from 750 to 0. The number of active PMs slightly decreases as the total number of small requests increases. The reason for this phenomenon is that the migration overhead falls with more small requests in the workload.

3.6 Conclusion

Motivated by various emerging memory-intensive applications in clouds, this chapter studied the resource management problem in order to better support the big memory applications in cloud environment. We designed a resource management algorithm named BigMem to reduce the resource usage of big memory jobs while satisfying good SLA to users. Specifically, we have developed a Markov Chain based analytical model for the

model-based provisioner. While the model-based provisioner performs its best effort to provision the needed active PMs, the overflow requests leaked from the provisioner due to variation of workload are further absorbed by the heuristic-based feedback control, which can be designed based on the committed SLA. Our developed model covers two unique memory provisioning features (migration and overcommit) and predicts the minimum number of PMs. Based on the real data in runtime, the model is able to adjust its strategies in provisioning to offer economical resources usage to cloud operators and acceptable SLA to users. We evaluate BigMem with synthetic workloads with basic and real-world patterns from Microsoft. Our experiments show that BigMem reveals effectiveness on optimizing resource usages in big memory clouds, which contributes on cost conservation and performance guarantee for both users and cloud operators.

The provisioning algorithm in this chapter is based on a basic scheduling method, first fit. In the future, more scheduling algorithm can be modeled and supported in BigMem. However, modeling a more complex scheduling method in a large scale data center is challenging, and how to overcome this issue will be the key contribution in the possible future work.

Chapter 4

Efficient Resource Allocation for Heterogeneous Workloads in IaaS Clouds

This chapter introduces the details of the resource allocation method for heterogeneous workloads, SAMR, in the following aspects: Section 4.1 introduces the backgrounds and motivations of this topic. Section 4.2 draws an overview of the proposed method; Section 4.3 discusses the resource allocation algorithm in SMAR including the new notation of VM offering, definition of multi-resource skewness and the resource allocation algorithm of SAMR; Section 4.4 introduces the resource prediction model based on Markov chain used in the resource allocation algorithm. In Section 4.5, we evaluate SAMR by simulations with both synthetic and real-world workloads. Section 4.6 concludes this chapter.

4.1 Introduction

In recent years, many efforts [5, 76, 7, 8, 9, 10, 13] have been devoted to the problem of resource management in IaaS public clouds such as Amazon EC2 [1] and Rackspace cloud [77]. All these works have shown their strength in some specific aspects in resource

scheduling and provisioning. However, existing works are all on the premise that cloud providers allocate VMs with homogeneous resource configurations. Specifically, homogeneous resource allocation offers resources in terms of VMs where all the resource types have the same share of the physical machine (PM) capacity. Both dominant resource and non-dominant resource are allocated with the same share in such manner even if the demands for different resources from a user are different.

Obviously, using homogeneous resource allocation approach to serve users with different demands on various resources is not efficient in terms of green and economical computing [14]. For instance, if users need Linux servers with 16 CPU cores but only 1GB memory, they still require to purchase *m4.4xlarge* (with 16 vCPU and 64 GB RAM) or *c4.4xlarge* (with 16 vCPU and 30 GB RAM) in Amazon EC2 [1] (July 2, 2015), or *Compute1-30* (with 16 vCPU and 30 GB RAM) or *I/O1-60* (with 16 vCPU and 60 GB RAM) in Rackspace [77] (July 2, 2015) to satisfy users' demands. In this case, large memory will be wasted. As the energy consumption by PMs in data centers and the corresponding cooling system is the largest portion of cloud costs [14, 18, 16], homogeneous resource allocation that provisions large amounts of idle resources wastes tremendous energy. Even in the most energy-efficient data centers, the idle physical resources may still contribute more than one half of the energy consumption in their peak loads. Besides, for cloud users, purchasing the appropriate amounts of resources for their practical demands is able to reduce their monetary costs, especially when the resource demands are mostly heterogeneous.

We observe that most resource demands of the applications in cloud workloads are diversified on multiple resource types (e.g., number of CPU cores, RAM size, disk size, bandwidth, etc.). As shown in Fig. 4.1, we analyzed the normalized resource (CPU and RAM) usages of a cloud computing trace from Google [78, 79] which consists of a large

4.1.a: Resource usage of CPU and RAM (normalized to (0, 1))

4.1.b: CDF of heterogeneity

Figure 4.1: Resource usage analysis of Google Cluster Traces.

amount of cloud computing jobs. It is clear that different jobs in Google trace have different demands in various resource types. Fig. 4.1.a shows the comparisons of normalized CPU and RAM usages for the first 1000 jobs in Google trace. We can see that most jobs do not utilize the same share of different resource types. Allocating resource according to the dominant resource naturally wastes many non-dominant resources. Fig. 4.1.b analyzes the distribution of the heterogeneity (defined as the difference between CPU and RAM usage, or $|CPU_{usage} - RAM_{usage}|$) for all jobs in Google trace. It reveals that more than 40% of the jobs are highly unbalanced between CPU and memory usage, and there are approximately 36% jobs with heterogeneity higher than 90%. Homogeneous resource allocation will not be cost-efficient for such heterogeneous workloads in the clouds because the non-dominant resources will be wasted significantly. Therefore, a flexible and economical resource allocation method for heterogeneous workloads is needed.

Nevertheless, consideration of heterogeneous workloads in resource allocation results in a number of challenges. Firstly, the resource demands in users' jobs are skewed among various resources. If the skewness of resource usages is ignored in resource allocation, some specific resource types with high demand may be exhausted before other resource

types with low demand. Secondly, the complexity of resource allocation considering multiple resource types will be significantly increased. The complexity of provisioning algorithms for homogeneous resource allocation [17, 20] is already high and the computational time is long given the large number of PMs in data centers nowadays. The further consideration of multiple resources adds additional dimensions to the computation which will significantly increase the complexity. Thirdly, the execution time of some jobs (e.g., Google trace) can be as short as a couple of minutes which rapidly changes the PM utilization. This rapid change makes provisioning and resource allocation challenging.

To cope with the heterogeneous workloads, we propose a skewness-avoidance multi-resource (SAMR) allocation algorithm to efficiently allocate heterogeneous workloads into PMs. SAMR designs a heterogeneous VM offering strategy that provides flexible VM types for heterogeneous workloads. To measure the skewness of multi-resource utilization in data center and reduce its impact, SAMR defines the multi-resource skewness factor as the metric that measures both the inner-node and the inter-node resource balancing. In resource allocation process, SAMR first predicts the required number of PMs under the predefined VM allocation delay constraint. Then SAMR schedules the VM requests using skewness factors to reduce both the inner-node resource balance between multiple resources and the inter-node resource balance between PMs in the data center. By such manner, the total number of PMs are reduced significantly while the resource skewness is also controlled to an acceptable level. Our experimental evaluation with both synthetic workloads and real world traces from Google shows that our approach is able to reduce the resource provisioning for cloud workloads by 45% and 11% on average compared with the single-dimensional method and the multi-resource allocation method without skewness consideration, respectively.

4.2 System overview

In this section, we introduce the application scenario of our research problem and provide a system overview on our proposed solution for heterogeneous resource allocation. Table 4.1 lists the key notations used throughout this chapter.

Table 4.1: Notations used in algorithms and models

K	Number of resource types
N_{total}	Total number of PMs in the considered data center
$\vec{R}$	r_i is the capacity of type- i resource in a PM, $i = [1, 2, \dots, K]$
$\vec{M}$	m_i , $m_i < r_i$ is the maximum resource for type- i resource in a VM, $i = [1, 2, \dots, K]$
X	Total number of VM types
$\vec{V}^x$	The resource configuration of type- x VM, v_i^x ($v_i^x \leq m_i$) represents the amount of type- i resource, $x = [1, 2, \dots, X]$
$\vec{C}$	c_i is the total consumed type- i resource in a PM, $c_i \leq r_i$, $i = [1, 2, \dots, K]$
$\vec{U}$	u_i is the utilization of type- i resource in a PM, $u_i \in [0, 1]$, $i = [1, 2, \dots, K]$
λ_x	Arrival rate of type- x requests, $x = [1, 2, \dots, X]$
μ_x	Service rate of type- x requests, $x = [1, 2, \dots, X]$
D	Predefined VM allocation delay threshold
d	Actual average VM allocation delay in a time slot
N	Provisioned number of active PMs (predicted by the model)
$\vec{S}$	s_n is the skewness factor for n^{th} active PM, $n = [1, 2, \dots, N]$

Similar to other works that optimize the resource usages in the clouds [14, 18, 16], we use the number of active PMs as the main metric to measure the degree of energy consumption in clouds. Reducing the number of active PMs in data center to serve the same amount of workloads with similar performance to users is of great attraction for cloud operators.

We consider the scenario where cloud users rent VMs from IaaS public clouds to run

Figure 4.2: System architecture of SAMR.

their applications in a pay-as-you-go manner. Cloud providers charge users according to the resource amounts and running time of VMs. Fig. 4.2 shows the system model of our proposed heterogeneous resource allocation approach SAMR. Generally, we assume that a data center with N_{total} PMs offers K different resource types (e.g., CPU, RAM, Disk, ...). Users submit their jobs as VM requests (also denoted as workloads in this chapter) with specific resource demands for different resource types, denoted as $q_1, q_2, \dots, q_K$. For each VM request, the cloud system attempts to allocate the most appropriate VM type that meets the user demands while minimizing the resource wastage. Thus, each job in the workloads will match one specific type of VM. We refer a job matching with type- x VM as a type- x job (or request). There are totally X VM types in the system. The resource configuration of type- x VM is expressed by a vector $\vec{V}^x = \{v_i^x | i = 1, 2, \dots, K\}$ which describes the resource amount of each resource type. All VM requests are maintained by a scheduling queue. For each request from users, resource scheduler allocates the resources for requested VM in N current active PMs if the resource slot of the VM is available. Otherwise, the request will be delayed waiting for more PMs to power up and join the service. According to the arrival rates and service rates of requests, SAMR conducts resource prediction based on a Markov Chain model periodically in every time

slot with a duration of t to satisfy the user experience in terms of VM allocation delay. By such manner, we focus on solving the problem in a small time period to increase the prediction accuracy. After the online prediction of required resources, the cloud system provisions corresponding number of active PMs N in the coming time slot. In job scheduling phase during each time slot with length t , cloud providers allocate resources and host each job into PMs using SAMR allocation algorithm.

In cloud service, one of the most significant impact on user experience is the service delay caused by schedulers. Here we consider the resource (or VM) allocation delay as the main metric for service-level-agreements (SLA) between users and cloud providers. Specifically, SAMR uses a VM allocation delay threshold D to be the maximum SLA value that cloud providers should comply with. Thus, there is a trade off between cost and SLA (as shown in Fig. 1.1) for cloud providers. To cope with the large amount of random request arrivals from users, it is important to provision enough active PMs. However, maintaining too many active PMs may cope well even under peak load but wastes energy unnecessary. Maintaining too few PMs may cause significant degradation in user experience due to lacks of active PMs and the need to wait for powering up more PMs. It is challenging to find the adequate number of active PMs. In our work, during the resource prediction phase, SAMR uses a Markov Chain model to find the adequate number of active PMs that satisfies the SLA value. Precisely, the model determines the number of active PMs, N , such that the average VM allocation delay d is smaller than the agreed threshold D .

We use the Markov Chain model to determine the adequate number of active PMs for operation. The model assumes heterogeneous workloads and balanced utilization of all types of resources within a PM. To realize the balanced utilization resources of different types, we define multi-resource skewness in PMs as a metric to measure the degree of unbalancing. The SAMR scheduling aims to minimize the skewness in data

center in order to avoid the resource starvation. The detail of skewness-avoidance resource allocation algorithm and model-based resource prediction are discussed in Section 4.3 and Section 4.4, respectively.

4.3 Skewness-Avoidance Multi-Resource allocation

In this section, we describe our proposed skewness-avoidance multi-resource allocation algorithm. Firstly, we introduce new notions of VM offering for heterogeneous workloads in clouds. Then we define skewness factor as the metric to characterize the skewness of the multiple resources in a PM. Finally, based on definition of skewness factor, we propose a SAMR allocation algorithm to reduce resource usage while maintaining the VM allocation delay experienced by users to a level not exceeding the predefined threshold.

4.3.1 New Notions of VM Offering

Generally, we consider a cloud data center with N_{total} PMs, each of which have K types of computing resources. We denote $\vec{R} = \langle r_1, r_2, \dots, r_K \rangle$ to be the vector describing the amount of each type of resources and $\vec{C} = \langle c_1, c_2, \dots, c_K \rangle$ to be the vector that describing the amount of resource used in a PM. To support better utilization of resources for heterogeneous jobs, it is necessary to consider a new VM offering package to cover the flexible resource allocation according to different resource types. We propose SAMR to offer a series of amounts for each resource type and allow arbitrary resource combinations that a user can pick. For instance, a cloud provider offers and charges VMs according to K resource types (e.g., CPU, RAM, disk storage, bandwidth,...) and the maximum amount of type- i resource ($i = 1, 2, \dots, K$, we refer i^{th} resource type as type- i resource in this chapter) is m_i . For each type of resource, there is a list of possible amounts for users to choose, and we consider a list of power of 2 for the amounts (e.g., 1, 2, 4, 8, ...).

Thus, the total number of types of VMs are $X = \prod_{i=1}^K (\log_2(m_i) + 1)$. We use $\vec{V}^x = \langle v_1, v_2, \dots, v_K \rangle$, for $x = [1, 2, \dots, X]$, to present a resource combination for type- x VM. SAMR allows users to select the suitable number of resource for each type. Thus, users are able to purchase the appropriate VMs that optimally satisfy their demands to avoid over-investments. We use an example to illustrate above VM offering package. A cloud system may identify two resource types: CPU and memory. The amounts of CPU (number of cores), memory (GB) are expressed by $\vec{V} = \langle v_1, v_2 \rangle$. If each PM have 16 CPU cores and 32 GB memory and it allows the maximum VM to use all the resources. Users can select 1 core, 2 cores, 4 cores, ..., or 16 cores of CPU combining with 1 GB, 2 GB, 4 GB, ..., or 32 GB of memory for their VMs. Thus, this configuration permits a total of 30 ($X = 30$) different types of VMs, namely $\langle 1, 1 \rangle^1, \langle 1, 2 \rangle^2, \dots, \langle 16, 16 \rangle^{29}, \langle 16, 32 \rangle^{30}$.

While the current virtualization platforms such as Xen and Openstack are ready to support this flexible offering, finding the right number of options to satisfy popular demands and developing attractive pricing plans that can ensure high profitability are not straightforward. We recognize that the precise design of a new VM offering is a complicated one. Our considered VM offering package is used to illustrate the effectiveness of SAMR. However, SAMR is not limited to a particular VM offering package.

4.3.2 Multi-Resource Skewness

As discussed in Section 4.1, heterogeneous workloads may cause starvation of resources if the workloads are not properly managed. Although live migration can be used to consolidate the resource utilization in data centers to unlock the wasted resources, live migration operations result in service interruption and additional energy consumption. SAMR avoids resource starvation by balancing the utilization of various resource types during the allocation. Migration could be used to further reduce the skewness in the runtime of cloud data center if necessary.

Skewness [18, 80] is widely used as a metric for quantifying the resource balancing of multiple resources. To better serve the heterogeneous workloads, we develop a new definition of skewness in SAMR, namely *skewness factor*.

Let $\mathbf{G} = \{1, 2, \dots, K\}$ be the set that carries all different resource types. We define the mean difference of the utilizations of K resource types as

$$Diff = \frac{\sum_{(i \in \mathbf{G}, j \in \mathbf{G}, i \neq j)} |u_i - u_j|}{K \cdot (K - 1)}, \quad (\text{Eq. 4.1})$$

where u_i is the utilization of i^{th} resource type in a PM. Then the average utilization of all resource types in a PM is $\bar{U}$, which can be calculated by

$$\bar{U} = \frac{\sum_{i=1}^K u_i}{K}. \quad (\text{Eq. 4.2})$$

The skewness factor of n^{th} PM in a cloud data center is defined by

$$s_n = \frac{Diff}{\bar{U}} = \frac{\sum_{(i \in \mathbf{G}, j \in \mathbf{G}, i \neq j)} |u_i - u_j|}{(K - 1) \cdot \sum_{i=1}^K u_i}. \quad (\text{Eq. 4.3})$$

The concept of skewness factor is denoted as a factor that quantifies the degree of skewness in resource utilization in a PM with multiple resources. The degree of skewness factor has the following implication and usages.

- The value of skewness factor is non-negative ($s_n \geq 0$), where 0 indicates that all different types of resources are utilized at the same level. The skewness factor closer to 0 reveals lower degree of unbalanced resource usages in a PM. Thus, our scheduling goal is to minimize the average skewness factor. In contrast, a larger skewness factor implies higher skewness, which means that the resource usages are skewed to some specific resource types. It also indicates that the PM has a high probability of resource starvation.

- The skewness factor is the main metric in skewness-avoidance resource allocation for heterogeneous workloads. Thus, in the definition of skewness factor, we consider two aspects of the characteristics of the resource usages in PMs to keep the inner-node and inter-node resource balancing. The first aspect is the mean differences between the utilizations of multi-resources within a PM, or inner-node aspect. A higher degree of difference leads to a higher skewness factor, which is translated to higher degree of unbalanced resource usage. The second aspect in skewness factor is the mean of utilization of multi-resources in a PM. When the first aspect, the mean difference, is identical in each PM in data center, SAMR always choose the PM with the lowest mean utilization to host new VM requests such that the inter-node balance between PMs is covered in the definition of skewness factor.
- The resource scheduler makes scheduling decisions according to the skewness factors of all active PMs in data center. For each VM request arrival, the scheduler calculates the skewness factor for each PM as if the VM request were hosted in the PM. Thus, the scheduler is able to find the PM with the most skewness reduction after hosting the VM request. This strategy not only keeps the mean skewness factor of the PM low, but also maintain a low mean skewness factor across PMs. The detailed operation of the skewness-avoidance resource allocation algorithm is provided in the next subsection.

4.3.3 Skewness-Avoidance Resource allocation

Based on the specification of the multi-resource skewness, we propose SAMR as the resource allocation algorithm to allocate heterogeneous workloads. Algorithm 3 outlines the operation of SAMR for each time slot of duration t .

At the beginning of a time slot, the system uses past statistics to predict the number of active PMs needed to serve the workloads. Our model-based prediction will be discussed

Algorithm 3 Allocation algorithm of SAMR

```

1: Provision  $N$  PMs with prediction model in Section 4.4
2: Let  $N'$  be the current number of PMs at the beginning of the time slot
3: if  $N > N'$  then
4:   Powering on  $N - N'$  PMs
5: else if  $N < N'$  then
6:   Shut down  $N' - N$  PMs
7: if a type- $x$  job arrives at cloud system with demand  $\vec{V}^x$  then
8:    $opt = 0$ 
9:    $s_{opt} = 0$ 
10:  for  $n = 1$  to  $N$  do
11:    if  $\vec{C} + \vec{V}^x \leq \vec{R}$  then
12:      Compute  $s_n$  with Eq. 4.3
13:      Compute new  $s'_n$  if host the type- $x$  request
14:      if  $s_n - s'_n > s_{opt}$  then
15:         $opt = n$ 
16:         $s_{opt} = s_n - s'_n$ 
17:  if  $opt == 0$  then
18:    Power on a PM to allocate the job
19:    Delay the VM allocation for time  $t_{power}$ 
20:     $N = N + 1$ 
21:  else
22:    Allocate this job to  $opt^{th}$  PM:  $\vec{C} = \vec{C} + \vec{V}^x$ 
23:  if a type- $x$  job finishes in the  $n^{th}$  PM then
24:    Recycle the resource:  $\vec{C} = \vec{C} - \vec{V}^x$ 

```

in detail in Section 4.4. Then, the system will proceed to add or remove active PMs based on the prediction.

As job request arrives, the system conducts the following steps: 1) The scheduler fetches one request from the job queue. According to the VM type requested by the job, the scheduler starts searching the active PM list for a suitable vacancy for the VM. 2) In the search of each PM, the scheduler first checks whether there are enough resources for the VM in the current active PM. If a PM has enough resources to host the requested VM, the scheduler calculates the new multi-resource skewness factor and records the PM with maximum decrease in skewness factor. For the PM without enough resources, the scheduler simply skips the calculation. 3) After the checking for all active PMs, the scheduler picks the PM with the most decrease in skewness factor to host the VM. The most decrease in skewness factor indicates the most improvement in balancing utilization of various resources. In the case that there is no available active PM to host the requested VM, an additional PM must be powered up to serve the VM. This request will experience additional delay (t_{power}) due to the waiting time for powering up the PM. 4) After each job finishes its execution, the system recycles the resources allocated to the job. These resources will become available immediately for new requests.

4.4 Resource Prediction Model

In this section, we introduce the resource prediction model of SAMR. The objective of the model is to provision the active number of PMs, N , at the beginning of each time slot. To form an analytical relationship between operational configurations and performance outcomes, we develop a Markov Chain model describing the evolution of resource usage for SAMR in the cloud data center. With the model, we can determine the optimal number of PMs for cost-effective provisioning while meeting VM allocation delay requirement.

One of the advantages of cloud computing is the cost effectiveness for users and service providers. Cloud users wish to have their jobs completed in the cloud in lowest possible cost. Therefore, reducing their cost by eliminating idle resources due to homogeneous resource provisioning is an effective approach. However, due to the complexity in multiple dimensional resource type management, large scale deployment of PMs, and the highly dynamic nature of workloads, it is a non-trivial task to predict the suitable number of active PMs that can meet the user requirement. Modeling all N_{total} PMs and all K types of resource in a data center leads to a model complexity level of $O((\prod_{i=1}^K r_i)^{3N_{total}})$ and $O((\prod_{i=1}^K r_i)^{2N_{total}})$ for computation and space complexity, respectively. For example, with 1000 PMs, 2 types of resources, each with 10 options, the system evolves over 10^{4000} different states. It is computationally intensive to solve a model involving such a huge number of states. Since the resources allocated to a VM must come from a single PM, we see an opportunity to utilize this feature for model simplification. Instead of considering all PMs simultaneously, we can develop a model to analyze each PM separately which significantly reduces the complexity.

We observe that the utilizations of different types of resources among different PMs in data center are similar in a long run under SAMR allocation algorithm because the essence of SAMR is keeping the utilizations balanced among different PMs. Since all active PMs share similar statistical behavior of the resource utilization, we focus on modeling a particular PM in the system. Such approximation method can largely reduce the complexity while providing an acceptable prediction precision. The model permits the determination of allocation delay given a particular number of active PMs, N . With the model, we propose a binary search to find the suitable number of active PMs such that the delay condition of $d \leq D$ can be met.

In our model, we first predict the workloads at the beginning of each time slot. There are many load prediction methods available in the literature [18, 72], we simply use the

Figure 4.3: State transitions in the model.

Exponential Weighted Moving Average (EWMA) in SAMR. EWMA is a common method used to predict an outcome based on past values. At a given time τ , the predicted value of a variable can be calculated by

$$E(\tau) = \alpha \cdot O_b(\tau) + (1 - \alpha) \cdot E(\tau - 1), \quad (\text{Eq. 4.4})$$

where $E(\tau)$ is the prediction value, $O_b(\tau)$ is the observed value at time τ , $E(\tau - 1)$ is the previous predicted value, and α is the weight.

Next, we introduce the details for modeling each PM in SAMR provisioning method. Similar to previous works [20, 16, 17], we assume that the arrival rate of each type of jobs follows Poisson distribution and the execution time follows Exponential distribution. For type- x VM, the arrival rate and service rate of a job in the workloads are λ_x and μ_x , respectively. Since we consider each PM separately, the arrival rate for one single PM is divided by N .

Let $\vec{C}$ (a K -dimensional vector) be the system state in Markov Chain model where c_i represents the total number of used type- i resource in a PM. We denote $T\{\vec{S}|\vec{C}\}$ to be the rate of transition from state $\{\vec{C}\}$ to state $\{\vec{S}\}$. The outward rate transition from a particular system state, $\vec{C}$, in our model is given in Fig. 4.3 where the evolution of the

system is mainly governed by job arrivals and departures. We provide the details of the state transitions in the following.

Let $I(\vec{C})$ be an indicator function defining the validity of a system state, where

$$I(\vec{C}) = \begin{cases} 1, & 0 \leq c_i \leq r_i, i = 1, 2, \dots, K \\ 0, & \text{otherwise.} \end{cases} \quad (\text{Eq. 4.5})$$

An allocation operation occurs when there is an arrival of VM request to the cloud data center. When a VM request for type- x VM demands for $\vec{V}^x$ ($\vec{V}^x \leq \vec{R}$) resources, the system evolves from a particular state $\vec{C}$ to a new state $\vec{C} + \vec{V}^x$ provided that $\vec{C} + \vec{V}^x$ is a valid state. The rate of such a transition is

$$T\{\vec{C} + \vec{V}^x | \vec{C}\} = \lambda_x \cdot I(\vec{C} + \vec{V}^x). \quad (\text{Eq. 4.6})$$

The release of resources occurs when a VM finishes its execution. The rate of a release operation is decided by the number of VMs of each types because different type of jobs have different execution time. The number of a particular type in service is proportionate to its utilization of the system. Let w_x be the number of type- x VMs in a PM, w_x can be computed by

$$w_x = \frac{\sum_{i=1}^K \left[\frac{\frac{\lambda_x v_i^x}{\mu_x}}{\sum_{z=1}^K \frac{\lambda_z v_i^z}{\mu_z}} \cdot c_i \right]}{K}, \quad (\text{Eq. 4.7})$$

where the number of type- x VMs is determined by the mean value of the number of type- x VM calculated by K different resource types. Upon a depart of a type- x request, the system state transits from state $\{\vec{C}\}$ to state $\{\vec{C} - \vec{V}^x\}$ with a transition rate given by the following

$$R\{\vec{C} - \vec{V}^x | \vec{C}\} = w_x \cdot \mu_x \cdot I(\vec{C} - \vec{V}^x). \quad (\text{Eq. 4.8})$$

With the above transition, the total number of valid states that the system can reach is expressed by

$$S = \prod_{i=1}^K (r_i + 1). \quad (\text{Eq. 4.9})$$

Then, an S -by- S infinitesimal generator matrix for the Markov Chain model (Q) can be constructed. The steady-state probability of each state, $p(\vec{C})$, can be solved numerically using the following balance equation,

$$p(\vec{C}) \cdot \left[\sum_{x=1}^X (w_x \cdot \mu_x \cdot I(\vec{C} - \vec{V}^x) + \lambda_x \cdot I(\vec{C} + \vec{V}^x)) \right] = \sum_{x=1}^X [p(\vec{C} - \vec{V}^x) \cdot \lambda_x \cdot I(\vec{C} - \vec{V}^x) I(\vec{C}) + p(\vec{C} + \vec{V}^x) \cdot w_x \cdot \mu_x \cdot I(\vec{C} + \vec{V}^x) I(\vec{C})]. \quad (\text{Eq. 4.10})$$

Obtaining the steady-state probabilities of the system allows us to study the performance at the system level. The resource utilization vector of a PM can be determined by

$$\vec{U} = \sum_{c_1=0}^{r_1} \sum_{c_2=0}^{r_2} \dots \sum_{c_K=0}^{r_K} p(\vec{C}) \cdot (\vec{C}/\vec{R}). \quad (\text{Eq. 4.11})$$

We now analyze the probability that a VM request is delayed due to under-provision of active PMs. Let P_{d_x} be the delay probability of type- x requests, it can be computed by

$$P_{d_x} = \sum_{c_1=0}^{r_1} \sum_{c_2=0}^{r_2} \dots \sum_{c_K=0}^{r_K} p(\vec{C}) \cdot (1 - I(\vec{C} + \vec{V}^x)) \quad (\text{Eq. 4.12})$$

The overall probability of a request being delayed in the considered time slot, P_d , can be determined by

$$P_d = \frac{\sum_{x=1}^X P_{d_x} \lambda_x}{\sum_{x=1}^X \lambda_x}. \quad (\text{Eq. 4.13})$$

After obtaining the above, the average VM allocation delay can be determined by

$$d = P_d \cdot J \cdot t_{power}, \quad (\text{Eq. 4.14})$$

where J is the total number of jobs and t_{power} is the time for powering up an inactive PM.

Model Complexity. The prediction model in SAMR uses a multi-dimensional Markov chain that considers the K types of resources simultaneously. The time complexity to obtain a solution for the model is $O((\prod_{i=1}^K r_i)^3)$ where the r_i is the capacity of i^{th} resource type. The space complexity of the model is $O((\prod_{i=1}^K r_i)^2)$ which is the size of the infinitesimal generator matrix. Based on the analysis, adding more resources to each PM contributes insignificant to the complexity, however it may trigger introduction of new VM options to the system which increases r_i as well as the computational time and space. Likewise, considering additional resource type will certainly add VM options which increases the computational time and space. Nevertheless, current cloud providers usually consider two ($K = 2$) or three ($K = 3$) resource types on offering VMs, and thus it remains practical for SAMR to produce the prediction of resource allocation scheme in real time.

PM Scalability. The number of PMs, N_{total} , influences the prediction model and VM allocation algorithm. In the prediction model, a binary search is needed to check for the suitable number of PMs. The complexity is $O(\log(N_{total}))$. For the VM allocation algorithm execution, as it performs linear check on each active PM, the complexity is $O(N_{total})$. The overall complexity of our solution is thus linear to the number of PMs.

4.5 Evaluation

In this section, we evaluate the effectiveness of our proposed heterogeneous resource allocation approach with simulation experiments. First, we introduce the experimental

setups including the simulator, methods for comparison and the heterogeneous workload data. Second, we validate SAMR with simulation results and then compare the results with other methods.

4.5.1 Experimental setup

Simulator. We simulate a IaaS public cloud system where VMs are offered in a on-demand manner. The simulator maintains the resource usage of PMs in the cloud and support leasing and releasing the resources for VMs requested by users. We consider offering of two resource types: CPU cores and memory. In our experiments, we set the time for powering on a PM to 30 seconds and the default average delay constraint is set to 10 seconds. The default maximum VM capacity is set to 32% of the normalized capacity of a PM. Besides, the default time slot for resource allocation is 60 minutes. To study their impact on system performance, sensitivities of these parameters are investigated in the experiments. We study the following performance metrics in each time slot: number of PMs per time slot, mean utilization of all active PMs, multi-resource skewness factor and average VM allocation delay. The number of PMs is the main metric which can impact the other three metrics.

Comparisons To evaluate the effectiveness of SAMR in serving highly heterogeneous cloud workloads, we simulate and compare the results of SAMR with the following methods: 1) single-dimensional (SD). SD is the basic homogeneous resource allocation approach that is used commonly in current IaaS clouds. Resource allocation in SD is according to the dominant resource, other resources have the same share of dominant resource regardless of users' demands. For scheduling policy, we simply choose first fit because different scheduling policies in SD have similar performance impact on resource usage. In first fit, the provisioned PMs are collected to form a list of active PMs and the order of PMs in the list is not critical. For each request, the scheduler searches the

list for available resources for the allocation. If the allocation is successful, the requested type of VM will be created. Otherwise, if there is no PM in the list that can offer adequate resources, this request will be delayed. 2) multi-resource (MR). Different from SD, MR is a heterogeneous resource allocation method which do not consider multi-resource skewness factor in resource allocation. MR offers flexible resource combinations among different types of resource to cover different user demands on different resource types. MR also uses first fit policy to host VMs in cloud data center. 3) optimal (OPT). An optimal resource allocation (OPT) is compared as the ideal provisioning method with oracle information of workloads. OPT assumes that all PMs run with utilizations of 100%. The provisioning results of OPT are calculated simply by dividing the total resource demands in each time slot by the capacity of the PMs. Thus, OPT is considered as the most extreme case that minimum number of PMs are provisioned for the workloads.

Workloads. Two kinds of workloads are utilized, synthetic workloads and real world cloud trace, in our experiments as shown in Fig. 4.4. In order to study the sensitivity of performance under different workload features, three synthetic workload patterns are used: growing, pulse and curve. By default, the lowest average request arrival rates of all three synthetic workload patterns are 1400 and the highest points are 2800. We keep the total resource demands of each type of jobs similar so that the number of jobs with higher resource demands is smaller. The service time of the jobs in synthetic workloads are set to exponential distribution with average value of 1 hour.

To validate the effectiveness of our methods, we also use a large scale cloud trace from Google which is generated by the logs from the large scale cloud computing cluster containing 11000 servers in Google company. The trace records the system logs during 29 days from May 2011 and we pick the logs in the first day of the third week for experiments. We extract 73905 job submissions, each of which contains the job starting time, running time, CPU usage and memory usage. The exact configurations of the servers in Google

4.4.a: Growing

4.4.b: Pulse

4.4.c: Curve

4.4.d: Google

Figure 4.4: Three synthetic workload patterns and one real world cloud trace from Google.

cluster are not given in the trace and the resource usages use normalized values from 0 to 1 (1 is the capacity of a PM). Thus we also use the normalized resource usages in experiments for both synthetic workloads and Google trace.

4.5.2 Experimental results

Overall results. We first present the overall results of the four methods for the four workloads. Fig. 4.5 shows the overall results for different metrics with all workloads

Figure 4.5: Overall results of four metrics under four workloads. The bars in the figure show average values and the red lines indicate 95% confidence intervals.

and resource management methods. The bars in the figure show the average values for different results and the vertical red lines indicate the 95% confidence intervals.

We make the following observations based on the results. Firstly, heterogeneous resource management methods (MR and SAMR) significantly reduce resources in terms of number of active PMs for the same workloads. As shown in Fig. 4.5(a), the resource conservation achieved by MR compared with SD is around 34% for all four workloads. SAMR further reduces the required number of PMs by another 11%, or around 45% compared with SD. It shows that SAMR is able to effectively reduce the resource usage by avoiding resource starvation in cloud data center. Besides, the number of active PMs for SAMR is quite close to the optimal solution with only 13% difference. Note that the presented number of active PMs for SAMR is the actual required number for the given workloads. Based on our experiment records, the predicted numbers of PMs from our model have no more than 5% (4.3% on average) error rates compared with the actual required numbers presented in the figure. Secondly, although the utilization of dominant

resource using SD method is high as shown in Fig. 4.5(b), the non-dominant resources are under-utilized. However, the resource utilizations in MR and SAMR policies are balanced. This is the reason that SD must provision more PMs. Thirdly, the effectiveness of resource allocation in SAMR is validated by the skewness factor shown in Fig. 4.5(c), where the average resource skewness factors in SAMR method are less than that in MR. Finally, all three policies achieve the predefined VM allocation delay threshold as shown in Fig. 4.5(d). SD holds slight higher average delays than SAMR and MR, which is due to the fact that SD always reacts slowly to the workload dynamicity and cause more under-provisioned cases to make the delay longer.

Impacts by the amount of workloads. Fig. 4.6 shows the detailed results of all methods for different metrics under four workloads. We highlight and analyze the following phenomenons in the results. Firstly, heterogeneous resource allocation methods significantly reduce the required number of PMs in each time slot for 4 workloads as in Fig. 4.6.a to Fig. 4.6.d. Secondly, from Fig. 4.6.e to Fig. 4.6.h we can see that SAMR is able to maintain high PM utilization in data center but the PM utilization of MR method fluctuates, falling down under 80% frequently. This is due to the starvation or unbalanced usage among multiple resource types in MR as shown in Fig. 4.6.i to Fig. 4.6.l. Thirdly, we observe that the utilization of CPU and RAM resources using SAMR are close in the three synthetic workloads but the difference in Google trace is large as shown in Fig. 4.6.e to Fig. 4.6.h. This is caused by the fact that the total demands of RAM is more than that of CPU in traces from Google Cluster. It can also be verified by the higher resource skewness factors in Fig. 4.6.i to Fig. 4.6.l, where the skewness factors in Google trace are much higher than the other three workloads.

We now perform sensitivity studies on major parameters. We investigate the impact of the system parameters including the degree of heterogeneity, delay threshold, the number of VM types and time slot length on the performance of multiple resource usage.

Figure 4.6: Detailed results of three metrics under four workload patterns.

For each experiment, we study the impact of varying one parameter while setting other parameters to their default values.

Impacts by workload heterogeneity. We first investigate the performance under different workload distributions with different degrees of heterogeneity. We run four experiments using Growing pattern in this study. In each experiment, the workload consists of only two types of VMs (the amounts of two types of VM are the same) with the same heterogeneity degree. Specifically, we use $\langle 1, 1 \rangle + \langle 1, 1 \rangle$, $\langle 1, 4 \rangle + \langle 4, 1 \rangle$,

Figure 4.7: Sensitivity studies for different degrees of heterogeneity (job distributions). The bars in the figure show average values and the red lines indicate 95% confidence intervals.

$\langle 1,8 \rangle + \langle 8,1 \rangle$, and $\langle 1,16 \rangle + \langle 16,1 \rangle$ in the first, second, third and fourth experiments, respectively. For all the experiments, we keep the total amounts of dominant resource identical in order to compare the impacts of heterogeneity on resource usage. Fig. 4.7 shows the results using SD, MR and SAMR with different heterogeneity. It can be seen that the required number of PMs increases as the heterogeneity increases in SD method but the number of PMs required in MR and SAMR falls with the increase of heterogeneity of the workloads. The reason is that large amounts of resources are wasted in SD, while MR and SAMR are capable to provide balanced utilization of resources. This phenomenon again shows the advantage of heterogeneous resource management for serving diversified workloads in IaaS clouds. The advantage becomes more obvious in SAMR which is specifically designed with skewness avoidance.

Impacts by different delay thresholds. Fig. 4.8(a) shows the results for varying the delay threshold D for Google trace. We use a set of delay threshold (minutes) : 15, 30, 60, 90, 120. We can see from the figure that the number of active PMs in each time slot reduces as we allow higher delay threshold. This is because a larger D value permits more jobs in the waiting queue for powering up additional PMs, and thus the cloud system is able to serve more jobs with the current active PMs. In practice, cloud

Figure 4.8: Sensitivity studies for delay threshold, maximum VM capacity and length of time slot using Google trace.

providers is able to set an appropriate D to achieve a good balance between quality of service and power consumption.

Impacts by maximum VM capacity. In Fig. 4.8(b), we design an experiment on Google trace where the cloud providers offer different maximum VM capacity. For example, a cloud system with the normalized maximum resource m_i offers $(\log_2 m_i \cdot 100 + 1)$ options on resource type- i . We test three maximum resource values 16%, 32%, 64%, respectively. From the figure we can see that with bigger VMs offered by providers, more PMs are needed to serve the same amount of workloads. The reason is that bigger VMs have higher chance to be delayed when the utilization of resources in the data center is high.

Impacts by time slot length. Fig. 4.8(c) shows the results for varying slot length from 15 minutes to 120 minutes using Google trace. Our heterogeneous resource management allows cloud providers to specify time slot according to their requirements. As shown in the figure, the number of active PMs can be further optimized with smaller time slots. These results suggest that we can obtain better optimization effect if our proposed prediction model and PM provisioning can be executed more frequently. However, the model computation overhead prohibits a time slot being too small.

4.6 Conclusion

Real world jobs often have different demands on different computing resources. Ignoring the differences in the current homogeneous resource allocation causes resource starvation on one type and wastage on other types. To reduce the monetary costs for users in IaaS clouds and wastage in computing resources for cloud system, this chapter first emphasized the need to have a flexible VM offering for VM requests with different resource demands on different resource types. We then proposed a heterogeneous resource allocation approach named skewness-avoidance multi-resource (SAMR) allocation. Our solution includes a VM allocation algorithm to ensure heterogeneous workloads are allocated appropriately to avoid skewed resource utilization in PMs, and a model-based approach to estimate the appropriate number of active PMs to operate SAMR. Particularly for our developed Markov Chain, we showed its relatively low complexity for practical operation and accurate estimation.

We conducted simulation experiments to test our proposed solution. We compared our solution with the single-dimensional method and the multi-resource method without skewness consideration. From the comparisons, we found that ignoring heterogeneity in the workloads led to huge wastage in resources. Specifically, by conducting simulation studies with three synthetic workloads and one cloud trace from Google, it revealed that our proposed allocation approach that is aware of heterogeneous VMs is able to significantly reduce the active PMs in data center, by 45% and 11% on average compared with single-dimensional and multi-resource schemes, respectively. We also showed that our solution maintained the allocation delay within the preset target.

This chapter addressed the problem of hosting heterogeneous workloads in homogeneous data centers. To extend this work, the homogeneous infrastructure can be considered in serving different type of workloads. It will be much more complex to model the

resource utilization in a heterogenous data center with different type of machines.

Chapter 5

QoS-aware Resource Allocation for Video Transcoding in Clouds

In this chapter, we introduce the resource allocation method for online video transcoding in clouds, COVT, in the following aspects: Section 5.1 draws the backgrounds and motivations of this work. Section 5.2 introduce the architecture of COVT including three different components. Section 5.3 introduces the profiling method in COVT. Section 5.4 derives the analytical model for the resource prediction with profiles of transcoding tasks in given infrastructure. The scheduling algorithm that dispatches video transcoding tasks into the cloud cluster with strict QoS constraints is discussed in Section 5.5. Section 5.6 implements a testbed of COVT and test the effectiveness of COVT by real data. To evaluate the effectiveness of COVT in large scale clusters, we simulate COVT system and run large data set in Section 5.7. Section 5.8 concludes this chapter.

5.1 Introduction

With the explosive growth of the demands for online video streaming services [81], video service providers face significant management problems on the network infrastructure and computing resources. As reported in [81], the world-wide video streaming traffic will

occupy approximately 69% of the total global network traffic in 2017. Therefore, the video data is becoming the “biggest” big data that contributes to a huge amount of IT investments such as networking, storage and computing. Besides, online real-time video streaming services such as online conferencing [38], live TV and video chat have been growing rapidly as the most important multimedia applications.

With the rapid growth of mobile market, increasing volumes of online videos are consumed by mobile devices. As a result, service providers often need to transcode the video contents into different video specifications (e.g., bit rate, resolution, quality, etc) with different QoS (e.g., delay, etc) for heterogeneous mobile devices, networks and user preferences. However, video transcoding [82, 83] is a time-consuming task, and how to guarantee acceptable QoS for large video data transcoding is very challenging, particularly for those real-time applications which hold strict delay requirement.

Cloud computing technology [50] holds many advantages on offering elastic and economical computing resource for online video applications. Compared to video service providers who invest on their own IT infrastructures, cloud-based video transcoding and streaming services are able to benefit from on-demand resource reservation, simpler system maintenance and lower investments. For service providers using their own data centers, they have to build up an infrastructure that satisfies QoS at the peak load. Such over-provisioning of resources is highly inefficient in terms of cost. In contrast, cloud-based transcoding systems only need to consider current workload amount and reserve suitable resources to offer predefined QoS.

Online transcoding of big-volume video contents in clouds brings new challenges. First of all, the key problem is that online video applications have strict delay requirement, which includes both transcoding delay and steaming delay. The steaming delay is largely determined by the targeted transcoded video size. Thus, guaranteeing small transcoding

delay as well as small targeted video sizes in cloud-based online video transcoding is crucial. The second challenge is the resource reservation strategy that balances resource cost and QoS. If the reserved resource is less than requirement, the video transcoding process in clouds will take long time, and thus the delay of video playback would be high. On the other hand, if provisioning too much resource, the unused resources are wasted. The third issue is brought by the heterogeneity of infrastructures. The transcoding time of video chunks is different on different physical servers. Thus, the hardware heterogeneity is an important factor that should be considered.

We propose a cloud-based online video transcoding system (COVT) to handle the above challenges. COVT focuses on resource provisioning and task scheduling in order to provide economical and QoS guaranteed cloud-based video transcoding. Our research goal is to minimize the amount of resource (in terms of the number of CPU cores) for online video transcoding tasks given specific QoS constraints. In particular, we consider two QoS parameters: *system delay* and *targeted chunk size*. System delay is defined as the time from the arrival of a video chunk to the completion of the transcoding, which consists of queuing time and transcoding time. The targeted chunk size is the average size of output video chunks, which is the key indicator for streaming overhead.

COVT performs performance profiling to obtain the transcoding time and the targeted chunk size for different transcoding modes under the specific hardware infrastructure. Based on the profiles, COVT designs a prediction model to analyze the relationship between QoS and the number of CPU cores. Besides, the model is capable to find the optimal distribution of different transcoding modes while minimizing the resource amount required for large volume video data. In the scheduling phase, COVT distributes the video transcoding tasks into the cloud cluster with a QoS guaranteed scheduling algorithm that dynamically reserves resources in clouds for dynamic transcoding workloads.

5.2 System architecture

Table 5.1: Notations used in the transcoding system

K	Number of video streams
L	Length of video chunks (seconds)
V	Number of video types
$\vec{B}$	$\vec{B} = \{b_v v = 1, 2, \dots, V\}$, b_v is the proportion of v^{th} video types and $\sum_{v=1}^V b_v = 1$
M	Number of transcoding modes
$\mathbf{T}$	Array of profiles for average transcoding time, $\mathbf{T} = \{t_v^m m = 1, 2, \dots, M, v = 1, 2, \dots, V\}$, t_v^m is the average transcoding time of video type v using m^{th} transcoding mode
$\mathbf{W}$	Array of profiles for average targeted video chunks size, $\mathbf{W} = \{w_v^m m = 1, 2, \dots, M, v = 1, 2, \dots, V\}$, w_v^m is the average size of video type v using m^{th} transcoding mode
$\vec{P}$	$\vec{P} = \{p_m m = 1, 2, \dots, M\}$, p_m indicates the probability that the system should use m^{th} mode and $\sum p_i = 1$
$\vec{O}$	$\vec{O} = \{o_m m = 1, 2, \dots, M\}$, o_m indicates the actual proportion of video chunks using m^{th} mode observed in system and $\sum o_i = 1$
N	Predicted number of CPU cores by probabilistic model
n	The reserved CPU cores in clouds
u	The actual number of CPU cores used in system, $u \leq n$
D_{max}	Constraint of average delay
d	Observed average delay in system
S_{max}	Constraint of average size for targeted video chunks
s	Observed average size of targeted video chunks in system

In this section, we introduce the system architecture and provide an overview of COVT. For better explanation, all the important notations and parameters used throughout this chapter are listed in Table 5.1.

Fig. 5.1 illustrates the system architecture of COVT which consists of three components: video consumer, video service provider and cloud cluster. Generally, video consumers request their favored videos from the service provider who is responsible to

stream the transcoded video contents to consumers. The service provider reserves and manages computing resources from clouds to comprise a transcoding cluster. The cloud cluster consists of a number of VMs that transcodes the source videos into targeted videos with a certain video specification (including format, resolution, quality, etc) with some QoS constraints. The service provider is charged according to the amount of resource reserved in clouds. The detailed description of the three components in COVT is discussed as follows.

Figure 5.1: System architecture of COVT.

5.2.1 Video consumer

The source videos (or workloads) to be transcoded and forwarded to customers are a number of streams of video data, each of which is partitioned into video chunks with L seconds playback length. The video consumer includes all kinds of devices such as personal computers, mobile phones, tablets and televisions that request video contents from the video service provider. For different terminals, the desirable videos in terms of data rate, resolution, format are different due to heterogeneous network bandwidth, hardware ability and software functions. The delay tolerance of the video service is

also different for different applications. For example, the online TV permits a relatively long delay of several minutes, but the delay for delay-sensitive applications such as online conferencing should usually be less than one or two seconds. Breaking the delay tolerance will result in poor playback quality in customers' devices.

We use two aspects of delay as the QoS constraints in COVT, namely *system delay* (d) and *targeted chunk size* (s). The system delay is defined as the time from the arrival of a video chunk to the completion of transcoding, which consists of queuing time and transcoding time. The targeted chunk size is the average size of output video chunks which is the key indicator for streaming time in networks (although video streaming is not a main concern in this chapter). We set two thresholds for system delay d and targeted video chunk size s as the QoS constraints that the system should comply with, denoted as D_{max} and S_{max} , respectively. The values of D_{max} and S_{max} are determined by the service provider according to the practical requirements of different applications.

5.2.2 Video service provider

On one hand, the video service provider is responsible for streaming the required targeted video contents to video consumers by reserving sufficient resources in clouds. On the other hand, the service provider tries to seek an economical solution for the transcoding system in order to save monetary costs. Thus, the service provider needs to find the optimal point in the trade off between costs and QoS. With these design goals, we introduce the system modules in the service provider including performance profiling, resource prediction and task scheduling as follows:

- We define the transcoding for a video chunk with L seconds playback length as a task in COVT. The performance profiling is a common way to obtain the performance of tasks in terms of transcoding time and targeted chunk size, which is

important to guide the resource reservation and the task scheduling. The performance profiling module considers to record the transcoding time and the targeted chunk sizes of different video types in the past with different transcoding modes under specific hardware. The concept of transcoding mode is a configuration that controls the compression ratio of the output video in video transcoding process. There are usually several transcoding modes that can be used for different system requirements. A faster mode means shorter transcoding time but a lower compression ratio or a larger chunk size. In contrast, a slower mode produces a smaller targeted chunk size with longer transcoding time. With the profiles, COVT is able to determine the suitable distribution of different transcoding modes for workloads and further reserve appropriate number of resources for given QoS. The details of the profiling method is given in Section 5.3.

- Resource provisioning is used to predict the number of resources that is needed for the workloads given predefined QoS constraints D_{max} and S_{max} . The resource provisioning in COVT is a general method that is feasible for different resource types. In this chapter, we use the number of CPU cores to be the units in resource provisioning. Other resource types (e.g., GPU) can be supported with specific profiling data for the considered resource types.

In resource prediction, we model the transcoding system in COVT as an $M/G/N$ queue with Poisson arrivals of video chunks produced from the video source. The service rates are determined by the profiles from the performance profiling module. By solving the queuing model, the QoS values d and s can be computed given the number of CPU cores N and the distribution of transcoding modes $\vec{P} = \{p_m | m = 1, 2, \dots, M\}$ ($\sum_{m=1}^M p_m = 1$). Then, it is feasible to find the minimum number of CPU cores by enumerating different transcoding modes. The detailed modeling process is introduced in Section 5.4.

- Task scheduling module is responsible for distributing the large number of video chunks into the cloud cluster for transcoding processing. The scheduling policy is based on the transcoding modes distribution $\vec{P}$ generated in the prediction model. Our basic idea is to use slower transcoding modes as much as possible as long as the system delay d meets the QoS constraint. Let $\vec{O} = \{o_m | m = 1, 2, \dots, M\}$ be the observed value for $\vec{P}$ in the scheduling phase. If the observed proportion of the slowest mode o_1 is less than the prediction p_1 too much, we increase the resource reservation for the subsequent time periods. On the other hand, if o_1 is greater than p_1 too much, we decrease the number of resource because there is space for optimizing. In this way, we are able to accommodate the mismatch between the prediction and the actual situations so as to minimize the cloud resource and guarantee the QoS constraints. The detailed scheduling algorithm will be discussed in Section 5.5.

5.2.3 Cloud cluster

Cloud cluster includes several working nodes (VMs) leased from the clouds which are responsible for transcoding the video chunks despatched to them and forwarding the targeted video chunks to video consumers in parallel. The service provider periodically reserves the resources from clouds according to the provisioning scheme obtained from the prediction model in Section 5.4 for given QoS constraints. In the runtime of the transcoding system, the service provider adjusts the reserved number of resources in the clouds with the scheduling algorithm discussed in Section 5.5 according to the instant states of the system. It is common that the predicted amount of resources mismatches the preset QoS constraints in the runtime, which will be well compensated by the scheduling algorithm. By such manner, COVT is able to strictly guarantee the preset QoS constraints for online video transcoding services.

Note that we use the number of CPU cores as the units for resource provisioning in clouds without loss of generality. For a given number of CPU cores N predicted by the model, how to reserve VMs from clouds (e.g., whether to lease two 4-core VMs or four 2-core VMs for the transcoding tasks requiring eight CPU cores) is determined by the specific pricing model in clouds, which is not in the scope of this chapter.

5.3 Performance Profiling

In this section, we introduce the performance profiling for the video transcoding system aiming to assist the resource prediction for a targeted video specification (format, resolution and quality). As we discussed in Section 5.2.2, in video transcoding, it is possible to produce different targeted video chunks with different sizes by using different transcoding modes. The design of different transcoding modes allows a flexible trade off between transcoding delay and targeted chunk size.

Generally, COVT recognizes M different transcoding modes (e.g., slow, medium, fast, ...) and V different video types (e.g., movie, news, sports, ...). We denote $\mathbf{T} = \{t_v^m | m = 1, 2, \dots, M, v = 1, 2, \dots, V\}$ and $\mathbf{W} = \{w_v^m | m = 1, 2, \dots, M, v = 1, 2, \dots, V\}$ to be the average transcoding time and the average output size of video chunks using the m^{th} transcoding mode for the v^{th} video type. We run all combinations of transcoding modes and video types to record the average transcoding time and output size of the history data (recent several hours or days). Then, the profiles obtained in the profiling ($\mathbf{T}$ and $\mathbf{W}$) are used as the input parameters for the prediction model.

Fig. 5.2 illustrates the relationship between the transcoding time and the output size with different transcoding modes (from slowest to fastest). We can see that the average processing time decreases as the transcoding mode varies from the slowest to the fastest, but the average output chunk size grows with the faster transcoding mode.

Figure 5.2: The relationship between transcoding modes and QoS.

Thus, there is a trade off between the processing time of transcoding tasks and the output video size. Since the system delay consists of transcoding time and queuing time, the transcoding time also contributes to the system delay. Therefore, the overall transcoding mode distribution needs to be located in an area where the conditions of $d \leq D_{max}$ and $s \leq S_{max}$ are satisfied. In next section, we discuss how to predict the minimum number of CPU cores that meet the conditions of QoS.

5.4 Resource Prediction Model

In this section, we introduce the prediction model for the given QoS requirement D_{max} and S_{max} . We formulate the problem using a queuing model and then develop an approximate solution for the proposed model.

5.4.1 Queuing model

In COVT, all the video chunks of the K video streams generated from the video source are maintained in a queue as shown in Fig. 5.3. We consider V video types and each video chunk belongs to one video type $v, v = 1, 2, \dots, V$, with a playback length of L

Figure 5.3: The queuing model in COVT.

seconds. We partition the operating time of the system into multiple time slots and the resource prediction is performed at the beginning of each time slot. In this section, we focus on the resource prediction for one single time slot. The two QoS parameters of the system are denoted by d and s for the average system delay and the output chunk size, respectively. The goal of the resource prediction model is to provision the minimum N that meets the QoS requirements of $d \leq D_{max}$ and $s \leq S_{max}$. The distribution vector of transcoding modes $\vec{P}$ is also determined by the model when obtaining the minimum N .

We model the video transcoding process in the system as an $M/G/N$ queue. Let l be the queue length that evolves with a video chunk arrival from a stream or a video transcoding task completion. A video chunk arrival at the queue increases the queue length by one, and the completion of processing a chunk decreases the queue length by one. The arrival rates follow Poisson distribution with an average value λ which is determined by the video generation speed in the video source and the number of streams. The service rates of the queuing system μ follow general distribution which are generated from the profiles obtained in the profiling module. Note that there are N CPU cores in system working in parallel, which means that the overall service rate in the model is μN .

In the queuing model, we study the relationship between the QoS values and different system settings including the transcoding mode distribution ($\vec{P}$), number of CPU cores (N) and the performance profiles ($\mathbf{T}$ and $\mathbf{W}$) that we have obtained from the profiling module. Denoting f and g as the functions for QoS values d and s , respectively. We

formulate the provisioning issue of COVT as the following optimization problem:

$$\min_{\vec{P}} N \quad (\text{Eq. 5.1})$$

$$s.t. \quad d \leq D_{max} \quad (\text{Eq. 5.2})$$

$$s \leq S_{max} \quad (\text{Eq. 5.3})$$

$$d = f(N, \vec{P}, \mathbf{T}) \quad (\text{Eq. 5.4})$$

$$s = g(\vec{P}, \mathbf{W}) \quad (\text{Eq. 5.5})$$

To solve the above problem, we must first derive the functions f and g . The derivation of function g is simpler than function f because g has no relation with the resource amount N .

Markov Chain is a common technique to solve queuing problems, but it is not suitable here. In Markov Chain, states are memoryless so that the transition from one state to another is independent from the other states. It means that inter-arrival and service time should be both exponentially distributed. However, the service time here follows general distribution in the $M/G/N$ queue in COVT. Thus, we cannot use Markov Chain to solve the model.

5.4.2 Solution

We observe that the delay of video chunks in the system can be divided into two parts: the waiting time in queue and the transcoding time in the cloud cluster, denoted as D_q and D_t , respectively. Thus, the average system delay of COVT d can be expressed as

$$E[d] = E[f(N, \vec{P}, \mathbf{T})] = E[D_q] + E[D_t], \quad (\text{Eq. 5.6})$$

where function E means the expectation function. Since $E[D_t]$ is just the average transcoding time of video chunks, we can get it through the profiles of transcoding time

T. Let μ be the average service (transcoding) rate of the queuing model which can be calculated by

$$\mu = \frac{1}{E[\mathbf{T}]} = \frac{1}{\sum_{m=1}^M p_m \cdot \sum_{v=1}^V b_v \cdot t_v^m}, \quad (\text{Eq. 5.7})$$

where $\vec{B} = \{b_v | v = 1, 2, \dots, V\}$ is the proportion of different video types and $\vec{P} = \{p_m | m = 1, 2, \dots, M\}$ is the distribution of transcoding modes. $E[\mathbf{T}]$ is the average transcoding time of a video chunk in one CPU core, which is computed over different video types and different transcoding modes. Thus, the overall service rate of the cluster is μN since there are N CPU cores in the system. Accordingly, the average processing time of a video chunk in the system with N CPU cores is given by $\frac{1}{\mu N}$. Then Eq. 5.6 can be written as

$$E[d] = E[D_q] + \frac{1}{\mu N}. \quad (\text{Eq. 5.8})$$

The queuing delay D_q also consists of two parts, the remaining processing time of current transcoding tasks in the cloud cluster and the sum of the transcoding time of all the chunks in the queue, i.e.

$$E[D_q] = E[R] + \frac{E[l]}{\mu N}, \quad (\text{Eq. 5.9})$$

where $E[R]$ stands for the remaining processing time of video chunks in the cloud cluster and $E[l]$ is the average queue length. With Little's formula

$$E[l] = \lambda E[D_q], \quad (\text{Eq. 5.10})$$

we obtain

$$E[D_q] = \frac{E[R]}{1 - \frac{\rho}{N}}, \quad (\text{Eq. 5.11})$$

where $\rho = \frac{\lambda}{\mu}$ is defined for convenience of expression. Therefore, Eq. 5.8 becomes

$$E[d] = \frac{E[R]}{1 - \frac{\rho}{N}} + \frac{1}{\mu N}. \quad (\text{Eq. 5.12})$$

Now, the issue is to derive $E[R]$. Since the remaining processing time of video chunks in COVT does not follow the exponential distribution and the memoryless property, we derive it from the beginning by considering all the tasks (chunks) in the cloud cluster.

Considering a long time interval $[0, Z]$, we denote $\Gamma(z), z \in [0, Z]$ to be the remaining processing time of video chunks in the cloud cluster at time z . Then we can calculate $E[R]$ by

$$E[R] = \frac{1}{Z} \int_0^Z \Gamma(z) dz. \quad (\text{Eq. 5.13})$$

Assume there are totally $I(Z)$ tasks arriving at the system in time interval $[0, Z]$. Then, let Y_i be the processing time of the $i^{\text{th}}, i = 1, 2, \dots, I(Z)$, transcoding task. To illustrate the processing time function $\Gamma(z)$ with the discrete video chunk arrivals, we show the evolving process in Fig. 5.4. As shown in the figure, the reminding processing time of $\Gamma(z)$ is equal to zero when there is no task in the cloud and set to Y_i as the task commences. Then the value of $\Gamma(z)$ decreases linearly with rate 1 till the task completion. It implies that the integration part in Eq. 5.13 is the sum of areas of all triangles under the curve $\Gamma(z)$, where the sides and heights of the triangles are both Y_i . Thus, for large Z , we derive

Figure 5.4: Processing time of tasks.

$$E[R] = \frac{1}{Z} \int_0^Z \Gamma(z) dz \quad (\text{Eq. 5.14})$$

$$= \frac{1}{Z} \sum_{i=1}^{I(Z)} \frac{1}{2} (Y_i)^2 \quad (\text{Eq. 5.15})$$

$$= \frac{I(Z)}{2Z} \cdot \frac{1}{I(Z)} \sum_{i=1}^{I(Z)} Y_i^2 \quad (\text{Eq. 5.16})$$

$$= \frac{1}{2} \lambda \overline{Y^2}, \quad (\text{Eq. 5.17})$$

where $\overline{Y^2}$ is the second moment of the processing time Y_i . With the relationship between variance and second moment

$$\sigma^2 = \overline{Y^2} - \frac{1}{(\mu N)^2}, \quad (\text{Eq. 5.18})$$

where the σ^2 is the variance of Y_i , we obtain

$$E[R] = \frac{1}{2} \lambda (\sigma^2 + \frac{1}{(\mu N)^2}), \quad (\text{Eq. 5.19})$$

where

$$\sigma^2 = \sum_{m=1}^M p_m \sum_{v=1}^V (\frac{b_v t_v^m}{N} - \frac{1}{(\mu N)})^2. \quad (\text{Eq. 5.20})$$

Finally, with Eq. 5.12 and Eq. 5.19, we derive the formula for the system delay $E[d]$ as

$$E[d] = \frac{N^2 \lambda^2 \sigma^2 + \rho^2}{2 \lambda N (N - \rho)} + \frac{1}{\mu N}. \quad (\text{Eq. 5.21})$$

The average targeted output size s is given by

$$E[s] = \sum_{m=1}^M p_m \sum_{v=1}^V b_v \cdot s_v^m. \quad (\text{Eq. 5.22})$$

After we derive the models for the QoS parameters, we are able to find the optimal resource reservation (N) with respect to the transcoding mode distribution ($\vec{P}$) as shown in Eq. 5.1. However, it is difficult to solve the problem with a close-form solution since there are multiple unknown variables in $\vec{P}$. We seek an approximation solution for

the optimization problem in Eq. 5.1, which is presented in Algorithm 4. In particular, considering the value of N must be an integer, we enumerate N starting from 1. For each p_m , we discretize the probability proportion by a gap $\tau, \tau < 1$, so that $p_m \in [0, \tau, 2\tau, \dots, 1]$. In searching for the solution, we use the rule of selecting as many slower modes as possible as long as the QoS constraints are satisfied. The benefit of such rule is to reduce the chunk size if the delay constraint is met.

Complexity Analysis. The complexity of Algorithm 4 is $O(N \cdot M^2/\tau)$, where N is provisioned number of CPU cores and M is the number of transcoding modes. Since N increases with the increased number of streams and M is a small positive integer, the complexity of the algorithm is quite low. Besides, the complexity is inversely proportional to the discretizing gap τ , for which we use a default value of 0.05 in our experiments.

5.5 Task scheduling

To ensure the QoS in the runtime, we develop a task scheduling algorithm that dispatches the tasks in the system queue to the cloud cluster based on the prediction of resource usage and transcoding mode distribution. It is inevitable that some mismatch exists between the predicted resource usage and the practical situation in the runtime due to dynamic workloads in the cloud-based transcoding system. Therefore, it is necessary to monitor and manage the QoS with dynamic adjustments in the task scheduling phase.

As shown in Fig. 5.5, the task scheduling function in COVT is responsible to distribute the video chunk at the top of the queue to be processed in the cloud, with consideration of the practical QoS values d and s . For each video chunk, the scheduler determines its transcoding mode by the principle of choosing the slower modes as much as possible. By such manner, the system transcodes the tasks with slower modes when the QoS values are very low and with faster modes when the QoS values are high (close to D_{max} or

Algorithm 4 Resource prediction in a time slot

Require:

T: Profiles of average transcoding time
W: Profiles of average chunk sizes
 D_{max} : QoS constraint of system delay
 S_{max} : QoS constraint of chunk size

Ensure:

$\vec{P}$: Predicted distribution of transcoding modes
 N : Predicted number of CPU cores
 d : Predicted average system delay
 s : Predicted average chunk size

- 1: $N = 0$
- 2: $d = -1$ and $s = -1$
- 3: **while** $d > D_{max}$ or $d < 0$ or $s > S_{max}$ or $s < 0$ **do**
- 4: $N = N + 1$
- 5: **for** $m = 1$ **to** $M - 1$ **do**
- 6: **if** $m == 1$ **then**
- 7: $p_m = 1$
- 8: **else**
- 9: $p_m = 1 - \sum_{i=1}^{m-1} p_i$
- 10: **for** $i = m + 1$ **to** M **do**
- 11: $p_i = 0$
- 12: **while** $p_m > 0$ **do**
- 13: $d = f(N, \vec{P}, \mathbf{T})$
- 14: $s = g(\vec{P}, \mathbf{W})$
- 15: **if** $d > D_{max}$ or $s > S_{max}$ **then**
- 16: $p_m = p_m - 0.05$ and $p_M = p_M + 0.05$
- 17: **else**
- 18: break

Figure 5.5: Illustration of video transcoding tasks scheduling.

S_{max}). After each task completion, the system records and updates the QoS values and the observed transcoding probability $\vec{O}$ which is the practical value for $\vec{P}$. Based on the observed value of the transcoding mode distribution, we can infer the utilization of CPU cores in the cluster. Then, we dynamically adjust the resource reservation in clouds to conserve costs while guaranteeing QoS.

The detailed scheduling algorithm is given in Algorithm 5. At the beginning of each time slot, the number of CPU cores reserved in clouds is set to the prediction N , and practical delay d , targeted video size s and observed transcoding mode distribution $\vec{O}$ are all set to zero. For each task j , we introduce the scheduling algorithm with the following steps.

Firstly, the system judges whether there is vacant CPU core in the cluster for the task in the queue top. If so ($u < n$), the system starts finding the suitable transcoding mode to process the task. The slower mode that satisfies the QoS requirements $d \leq D_{max}$, $s \leq S_{max}$ is used for the task in scheduling.

Secondly, if there is no available CPU core immediately for the task, the system checks whether o_1 is within a reasonable range specified by THR , where o_1 is the practical

Algorithm 5 Task scheduling in a time slot

Require:

- T**: Profiles of average transcoding time
- W**: Profiles of average chunk sizes
- $\vec{P}$: Predicted distribution of transcoding modes
- N : Predicted number of CPU cores in each time slot

Ensure:

- n : Provisioning result of the system
 - d : Actual average system delay
 - s : Actual average chunk size
 - 1: $u = 0$ //The number of used CPU
 - 2: $j = 0$ //Task number
 - 3: $\vec{O} = \vec{0}$ //Observed proportion of transcoding modes
 - 4: $d = 0, s = 0$
 - 5: Let v_j be the video type of task j , $v_j = 1, 2, \dots, V$
 - 6: Let α_j^m be the practical transcoding time of chunk j using m^{th} mode
 - 7: Let β_j^m be the practical output size of chunk j using m^{th} mode
 - 8: **for** each time slot **do**
 - 9: $n = N$
 - 10: **for** task j in the system **do**
 - 11: **if** $u < n$ **then**
 - 12: **for** $m = 1$ **to** M **do**
 - 13: **if** $(w_{v_j}^m + s \cdot j)/(j + 1) \leq S_{max}$ and $(t_{v_j}^m + d \cdot j)/(j + 1) \leq D_{max}$ **then**
 - 14: Transcode j with mode m
 - 15: **else**
 - 16: **if** $(1 - THR) \cdot p_1 \leq o_1 \leq (1 + THR) \cdot p_1$ **then**
 - 17: Wait for a while and $m = M$
 - 18: **else**
 - 19: Reserve one more CPU core in the cloud
 - 20: $n = n + 1$
 - 21: $m = M$
 - 22: Transcode j with mode M
 - 23: $s = (\beta_j^m + s \cdot j)/(j + 1)$
 - 24: $d = (D_q + \alpha_j^m + d \cdot j)/(j + 1)$
 - 25: $u = u + 1$
 - 26: **if** $m == 1$ **then**
 - 27: $o_1 = (o_1 \cdot j + 1)/(j + 1)$
 - 28: **else**
 - 29: $o_1 = (o_1 \cdot j)/(j + 1)$
 - 30: **if** $o_1 < (1 - THR) \cdot p_1$ **then**
 - 31: $n = n + 1$
 - 32: **else if** $o_1 > (1 + THR) \cdot p_1$ **then**
 - 33: $n = n - 1$
 - 34: $j = j + 1$
 - 35: **for** a video chunk is finished transcoding **do**
 - 36: $u = u - 1$
-

proportion of the slowest mode used in the system and THR , $THR < 1$, is a preset threshold. $(1 - THR) \cdot p_1 < o_1 < (1 + THR) \cdot p_1$ means that the actual number of tasks using the slowest mode is neither too high nor too low. Thus, the lack of available CPU core is a temporary situation, and the system will let the task wait for sometime for available CPU core. But if o_1 is not in the range and there is no available CPU core, the system will reserve one more CPU core in the cloud to alleviate the high utilization of resource to guarantee QoS. Then, the task in queue top will be processed with M^{th} (fastest) mode.

Thirdly, after the processing, the system updates the records of practical QoS values as well as the number of CPU cores utilized. Besides, the proportion of the slowest transcoding mode used in the system is also updated as an important indicator for resource utilization. Every time when the task in queue top is processed with the slowest mode, the system increases the value of o_1 . On contrast, the system decreases the value of o_1 when the task is not processed with mode 1. Then, if o_1 is larger than $(1 + THR) \cdot p_1$, the system will reduce the number of CPU cores in order to save cost because most of tasks are processed by the slowest mode. If o_1 falls below $(1 - THR) \cdot p_1$, the system adds one CPU core to meet the computing needs. By such manner, COVT is capable to dynamically reserve resources in clouds under different system states and strict QoS.

5.6 Testbed Experiments

5.6.1 Experiment setup

We implement a prototype of COVT and evaluate its performance on a cluster with six VMs that are hosted on a server with a six-core Xeon E5-1650 CPU and 16GB DRAM. Each VM is a transcoding worker (with one CPU core and two GB memory) that runs the transcoding algorithm for video chunks. Besides, we deploy another server as the

video service provider that is responsible for making resource scheduling decision and communicating with the cloud cluster. The whole system is built by python and the transcoder we use for video transcoding is an efficient video processing tool, FFmpeg. For convenience, we utilize two transcoding modes ($M = 2$) in the prototype system of COVT, namely *fast* and *slow* modes (note that they correspond to ultrafast and veryslow in FFmpeg, respectively). We consider four video types ($V = 4$) including movie, news, advertisement (AD) and sport. The threshold factor THR is set by default to 0.1. The default QoS constraints of delay and output size are 2 seconds and 500 KB, respectively.

Figure 5.6: Workloads in experiments.

We use four video streams as the workloads for the cloud cluster in the experiments as shown in Fig. 5.6. The video data in streams 1 and 2 are a soccer game in World Cup 2014 and a table tennis game in Olympic game 2012, respectively. The data for stream 3 and 4 are from the famous TV station *Phoenix TV* in Hongkong. To show the performance under dynamic workloads, the streams are with different starting time

and finishing time. We partition the total operating time into time slots with a length of 30 minutes and the resource provisioning is performed for each time slot. All the video contents are segmented into 5 ($L = 5$) seconds chunks in playback length with the container of MP4 and the resolution of 640x360. The videos are transcoded to H.264 with the container of AVI and the resolution of 320x240.

We compare COVT with two other schemes: peak-load provisioning (Peak-load) and heuristic provisioning (Heuristic). In Peak-load, it always reserves the number of resource that satisfies the QoS in the peak loads. For Heuristic provisioning method, it adopts purely on-demand method to allocate the required resource for workloads. Specifically, Heuristic increases resource when there is no available resource for tasks in the runtime and decreases resource in case that the utilization of provisioned resource is too low (we use 70% as the threshold for the utilization rate).

5.6.2 Experimental Results

Profiling results. We firstly consider to obtain the profiles of the transcoding time and the targeted video chunk size. By running the video data one hour prior to the workloads in Fig. 5.6, we record the average transcoding time and the video chunk size for different video types and transcoding modes under the considered infrastructure, which are illustrated in Fig. 5.7. The bars represent average values and the red vertical lines show the corresponding 95% confidence intervals.

From Fig. 5.7, it can be seen that the transcoding time and the chunk size are significantly different for different transcoding modes. Specifically, the time for transcoding a video chunk using the slow mode is nearly 20 times of that using the fast mode, which offers a large space for the service provider to schedule the resource for a predefined QoS goal. Besides, the processing time of transcoding tasks for different types of video

Figure 5.7: Profiling results with two transcoding modes and four video types.

are closer using the fast mode. The case for the slow mode is more complicated since it depends on the data content. The average size of chunks with the fast mode is approximately triple of that with the slow mode. Thus, the slow mode produces smaller targeted video chunk size than the fast mode but takes longer transcoding time. Based on these profiling data on CPU cores under our experimental environment, COVT is able to predict the suitable number of cores for the workload.

Overall comparisons. Next, we present the overall comparisons of COVT with other methods in terms of resource provisioning for the online transcoding workloads in Fig. 5.8, where the provisioned numbers of CPU cores for the four hour period for Peak-load, Heuristic, model prediction and our proposed COVT are illustrated. We can see that the results of the prediction and COVT are quite close while the Heuristic approach differs in the beginning with climbing number of resources and in the end with falling resource provisioning. This is due to the deficiency of Heuristic that reacts slowly to the dynamic variation of workloads. Overall, COVT is able to conserve 25% resources in terms of CPU-hour compared with Peak-load for the workloads.

Together with Fig. 5.9 which draws the detailed information in the system runtime, we can see that Heuristic approach cannot meet the QoS requirements since it only passively reacts to the dynamic workloads. For example, at the beginning of the workloads,

Figure 5.8: Comparison of resource provisioning for different methods.

the provisioned CPU cores are lesser but the delay QoS is broken in Heuristic approach in Fig. 5.9 (b). Similar QoS broken can be viewed in Fig. 5.9 (c). In contrast, the results of COVT comply with the QoS constraints strictly, which demonstrates the effectiveness of COVT in provisioning QoS-sensitive video transcoding services. The 95% confidence intervals (8 time slots) of the results of COVT over multiple tests in Fig. 5.9 (a)

Figure 5.9: Detailed results of slow mode proportion, delay and chunk size.

Figure 5.10: Parameter studies of the testbed experiments.

, Fig. 5.9 (b) , Fig. 5.9 (c) are $[0.012, 0.018, 0.021, 0.023, 0.016, 0.019, 0.022, 0.018]$, $[0.043, 0.037, 0.029, 0.045, 0.056, 0.043, 0.049, 0.053]$ and $[167, 203, 151, 214, 176, 208, 142, 128]$, respectively.

Impacts of delay constraint. Fig. 5.10 (a) shows the parameter study of the delay constraint D_{max} . Fixing other parameters to their default values, we run the experiments with a set of D_{max} in $[1, 2, 3, 4]$ seconds. From the figure, we can see that the number of CPU cores reserved for the same workloads decreases when the delay constraint D_{max} increases. As the QoS constraint gets looser, less resource is needed to meet the delay requirement.

Impacts of chunk size constraint. The targeted chunk size is studied in Fig. 5.10 (b). Similarly, the required number of CPU cores decreases as S_{max} increases. The studies imply that both QoS parameters have a significant influence on the cloud resource and thus they need to be carefully selected based on specific application requirements.

Impacts of threshold factor. Fig. 5.10 (c) gives the results for varying the threshold factor THR in the task scheduling algorithm. As shown, a larger THR (0.3) reacts less frequently than a smaller one (0.1). A small THR can adjust the number of resource

Figure 5.11: Simulation results for large scale data set.

quickly according to bursty in workloads. On the other hand, it should be limited by the minimum resource reservation period in the clouds.

5.7 Simulation Evaluation

The results of the testbed experiments reveal the effectiveness of COVT on provisioning online transcoding services with practical system setup in a small virtual cluster. In this section, we seek simulation studies for COVT in order to investigate the scalability beyond the limitation of the scale of the testbed infrastructure.

We develop discrete-time event simulation with simulated workloads. We consider four workloads with 10, 50, 100, 200 as the maximum number of streams (also total time slots). For each workload, the number of streams increases from 1 in the first time slot to the maximum number with an increment 1. The simulation shares the same settings with the testbed experiments including the two transcoding modes and four video types as well as the profiling data. The proportion of different video types are equal in each time slot. The default QoS constraints of delay and output size are set to 1 second and 500 KB, respectively.

Simulation results. The simulation results with large scale data set are illustrated

in Fig. 5.11. Based on the results of provisioned number of CPU cores under different workload scales for COVT and Peak-load approach in Fig. 5.11 (a), we make the following observations.

Firstly, the results reveal that our proposed system COVT is capable to work for large scale online video transcoding in clouds. As the workloads vary from 1 in the first time slot to 200 streams in the 200th slot, COVT precisely provisions appropriate number of CPU cores for the predefined QoS.

Secondly, the advantage of using clouds as the processing platform is validated. As shown, the actual resource requirements of COVT is significantly less than Peak-load by approximately 47%. Thus, Peak-load solution for online video transcoding would waste a lot of investments on the IT infrastructure.

Figure 5.12: Number of CPU cores per stream.

Thirdly, it is observed that the resource conservation rate is higher when the maximum number of video streams gets larger. It is more clearly illustrated in Fig. 5.12, which shows the number of CPU cores required for each video stream in the workloads, namely CPU

per stream (CPS). CPS varies from greater than one in the testbed experiments to one in simulation using 10 streams and further decreases to less than one (approximately 0.63) when there are 200 streams. It implies that the larger the scale of the data set, the more benefit we can have for the proposed cloud based video transcoding.

Fig 5.11 (b) and Fig 5.11 (c) investigate the impacts of the QoS values D_{max} and S_{max} on the resource provisioning under large scale data set. They show stable effectiveness on different scales of workloads.

Overall, the simulation results validate the effectiveness of COVT for large scale online video transcoding applications. It shows the advantages in efficient resource conservation, elastic computing, QoS guarantee and high scalability.

5.8 Conclusion

In this chapter, we have proposed a novel method called COVT to address the problem of economical and QoS-aware online video transcoding in cloud environment. COVT considers two types of QoS: system delay and targeted video chunk size. As the transcoding time and the targeted chunk size are different on different hardware using different transcoding modes, we perform profiling for the cloud cluster to assist resource prediction and scheduling. We partition video streams into video chunks with short playback time and schedule them as tasks in clouds. We model the video streams as a queue and derive the relationship between QoS values and number of resources, based on which we solve the minimum number of resources with the QoS constraints. With the resource prediction, we have also proposed a scheduling algorithm to schedule transcoding tasks into clouds with strict QoS guarantee.

We perform both testbed and simulation experiments to evaluate our method on real-world workloads and large-scale simulated workloads, respectively. The results show that

our proposed solution is effective in terms of resource provisioning compared with the method that provisions physical resource according to the peak load. The number of CPU cores using COVT is up to 47% less than Peak-load in our experiments. It also showed that COVT outperforms Heuristic approach in terms of QoS guarantee. Overall, the experimental results validate our design goal that provisions minimum amount of resource while satisfying QoS constraints.

A possible future work is to develop a whole cloud-based video streaming service that includes both video transcoding and streaming. The resource required in each phase could be scheduled together. Such service allows online real-time applications with very short delay QoS and good user experience.

Chapter 6

Conclusion and Future Works

6.1 Conclusion

This thesis studied three topics in the direction of resource allocation in clouds. The key issue of resource allocation in IaaS clouds is balancing the trade off between the amount of resource provisioned and the QoS of cloud applications. How to determine a suitable amount of resource and schedule VMs or workloads into clouds are the two key problems. Thus, all these three topics are under the theme of cost-effective and QoS-aware resource management in clouds.

Firstly, motivated by the fact that current resource allocation methods in clouds almost target at CPU based applications and have no support for memory-intensive applications, a provisioning method for big memory clouds is proposed with consideration of VM migration and resource overcommit. The proposed method fully considers the features of memory allocation so as to reduce the total resource provisioned in clouds for big memory applications with acceptable VM allocation delay.

Secondly, since a resource management method for memory-intensive applications has been proposed, a problem that how to manage the resource in clouds for different type of applications comes out. This motivates us to study a resource allocation method

for heterogeneous workloads in order to reduce the resource consumptions in clouds. To characterize the resource imbalance in data centers with heterogeneous VMs, the skewness factor is proposed, which not only reflects the imbalance of resource usages among PMs in the data center, but also indicates the imbalance among different resource types in a PM. Based on the skewness factor, a resource prediction and a scheduling algorithm are designed to offer minimum number of PMs for heterogeneous workloads.

Lastly, after the investigation of the two resource allocation problems for general workloads, we turn to a specific resource allocation problem in a higher level of cloud computing workloads which is cloud-based video transcoding. The purpose is to catch the unique performance requirement of video transcoding services in resource allocation. To precisely predict the provisioning amount of CPU cores, we use a profiling method to get the performance of transcoding tasks for different video type under specific infrastructures. Based on the profiles, a model based provisioning and a heuristic scheduling algorithm is proposed. Since high delay is not permitted in video streaming services, our proposed cloud-based online video transcoding system always holds strict constraints on delay when provisioning the required number of CPU cores for workloads and scheduling transcoding tasks into the cloud cluster.

Evaluations by simulation and testbed experiments have shown that these methods effectively achieved our preset goal that reduces the resource provisioning in clouds without compromising QoS (or SLA) requirements. This result can be translated to a systematical resource management framework that includes cost-effective and QoS-aware resource allocation methods in different levels in cloud workloads: homogeneous workloads, heterogeneous workloads and specific applications.

6.2 Future Directions

6.2.1 Extension of Resource Allocation in Clouds

In this subsection, we discuss some possible research directions in the field of resource allocation in clouds. They are also possible extensions of our works in this thesis. The methods proposed in this thesis are capable to provide appropriate QoS or SLA to cloud users to keep an acceptable experience with low cost. Though these methods are efficient in offering cost-effective and QoS-aware resource allocation, there are still some promising directions in this topic to further improve the resource allocation efficiency. We summarize the following five possible future directions.

Firstly, BigMem proposed a resource allocation method for cloud operators to support memory-intensive applications in clouds with the VM as the allocation unit. However, it is common that a job needs a cluster of VMs. It is meaningful to design a method that allocates resources for a cluster of VMs together in clouds. Since all VMs in a cluster may not be often hosted in one single PM, the consideration of the cluster allocation will be a challenging task.

Secondly, SAMR proposed a new VM offering scheme with arbitrary sized VMs of different resource types and a skewness factor to characterize the imbalance of resource utilizations, which only focuses on guiding resource allocation in clouds. However, one important issue that we did not consider for heterogeneous resource allocation is the pricing model. Since different resource types are provisioned separately, a dynamic pricing model would help cloud operators to improve profit. For example, a resource type with low utilization in the data center may be charged a bit lesser than other resource types to incentive more purchases of that type of resource.

Thirdly, COVT focuses on the resource allocation in clouds, but ignores the pricing model for transcoding services. The video service provider should design a pricing model

for all different video specifications (e.g., format, quality, resolution, etc) and QoS requirements (e.g., delay). With the consideration of pricing model for cloud-based video transcoding, the service providers can adjust their resource reservation according to their profit, which is more practical in real systems.

Fourthly, the research works in this thesis only consider different workloads. How to schedule different workloads in a heterogeneous infrastructure is also a practical challenge for cloud computing. The difficulty of modeling both different types of workloads and hardware with different capacities increases significantly.

Fifthly, different applications have different QoS requirements, which requires different resource allocation schemes to provide cost-effective and QoS-aware resource management. It is interesting to study the resource allocation problems for more specific applications (such as big data processing and distributed machine learning, etc) to optimize their resource usages.

6.2.2 Other Research Issues in Cloud Computing

In this subsection, we will discuss some possible research directions beyond resource allocation problem in clouds. These topics are general research problems in clouds from our opinion, which are summarized as following two points.

The first interesting research topic in cloud computing is the support for big data applications. In the development process of cloud computing technology, lots of basic supporting techniques are developed and improved such as data center architecture, resource and energy management, virtualization, security, data center networking and pricing model, etc. Though these problems addressed the basic problems of cloud architecture, there is no prominent contribution for any new application. Cloud computing only offers a different computing architecture. As many big data applications get popular, cloud computing is able to create new data-driven services by supporting an entire

solution for big data applications including data uploading, storage, backup and security, processing framework and a data-aware pricing strategy.

The second research issue is the mobile clouds that deploy services in the base-stations of mobile networks to enhance performance of existing applications as well as to create new applications or services. With the advantage of mobile cloud computing like low latency, location-aware and wireless connection, a lot of applications that cannot achieve acceptable performance with current mobile networks such as mobile internet games and location-aware services are able to obtain significantly improvements. Meanwhile, there are a lot of research issues in this topic to be addressed by academia such as service stability in wireless environment, location-aware services, deployment strategy, pricing model and mobility management.

Author's Publications and Submissions

Journal Papers

- (i) **Lei Wei**, Chuan Heng Foh, Bingsheng He and Jianfei Cai, “Towards Efficient Resource Allocation for Heterogeneous Workloads in IaaS Clouds”, *Cloud computing, IEEE Transaction of*, 2015.
- (ii) **Lei Wei**, Jianfei Cai, Chuan Heng Foh and Bingsheng He, “QoS-aware Resource Allocation for Video Transcoding in Clouds”, *Circuits and systems for video technology, IEEE Transaction of*, under second round review, 2015.
- (iii) **Lei Wei**, Chuan Heng Foh, Bingsheng He and Jianfei Cai, “BigMem: Efficient Resource Management for Memory-Intensive Applications in Clouds”, *Big Data, IEEE Transaction of*, under review, 2016.

Conference Papers

- (i) **Lei Wei**, Bingsheng He and Chuan Heng Foh, “Towards Multi-Resource Physical Machine Provisioning for IaaS Clouds”, in *Proceedings of IEEE International Conference on Communications (ICC)*, Sydney, Australia, June 2014.

References

- [1] Amazon Pricing, <https://aws.amazon.com/ec2/pricing/>.
- [2] D. Villegas, A. Antoniou, S. M. Sadjadi, and A. Iosup, “An analysis of provisioning and allocation policies for infrastructure-as-a-service clouds,” in *Proc. of 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid’12)*. IEEE, 2012, pp. 612–619.
- [3] K. Mills, J. Filliben, and C. Dabrowski, “Comparing vm-placement algorithms for on-demand clouds,” in *Proc. of CLOUDCOM’11*, 2011.
- [4] E. G. Coffman Jr, M. R. Garey, and D. S. Johnson, “Approximation algorithms for bin packing: A survey,” in *Approximation algorithms for NP-hard problems*. PWS Publishing Co., 1996, pp. 46–93.
- [5] S. Genaud and J. Gossa, “Cost-wait trade-offs in client-side resource provisioning with elastic clouds,” in *Proc. of 2011 IEEE International Conference on Cloud Computing (CLOUD’10)*. IEEE, 2011, pp. 1–8.
- [6] D. Xie, N. Ding, Y. C. Hu, and R. Kompella, “The only constant is change: incorporating time-varying network reservations in data centers,” *ACM SIGCOMM Computer Communication Review*, pp. 199–210, 2012.
- [7] P. Marshall, H. Tufo, and K. Keahey, “Provisioning policies for elastic computing environments,” in *Proc. of 2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW)*. IEEE, 2012, pp. 1085–1094.

REFERENCES

- [8] L. Wang, J. Zhan, W. Shi, and Y. Liang, “In cloud, can scientific communities benefit from the economies of scale?” *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, no. 2, pp. 296–303, 2012.
- [9] R. V. den Bossche, K. Vanmechelen, and J. Broeckhove, “Cost-optimal scheduling in hybrid iaas clouds for deadline constrained workloads,” in *IEEE CLOUD’10*, 2010.
- [10] M. Malawski, G. Juve, E. Deelman, and J. Nabrzyski, “Cost-and deadline-constrained provisioning for scientific workflow ensembles in iaas clouds,” in *Proc. of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC’12)*. IEEE Computer Society Press, 2012, p. 22.
- [11] A. Ali-Eldin, M. Kihl, J. Tordsson, and E. Elmroth, “Efficient provisioning of bursty scientific workloads on the cloud using adaptive elasticity control,” in *Proc. of the 3rd workshop on Scientific Cloud Computing Date*. ACM, 2012, pp. 31–40.
- [12] S. Niu, J. Zhai, X. Ma, X. Tang, and W. Chen, “Cost-effective cloud hpc resource provisioning by building semi-elastic virtual clusters,” in *Proc. of International Conference for High Performance Computing, Networking, Storage and Analysis (SC’13)*. ACM, 2013, p. 56.
- [13] K. Deng, J. Song, K. Ren, and A. Iosup, “Exploring portfolio scheduling for long-term execution of scientific workloads in iaas clouds,” in *Proceedings of International Conference for High Performance Computing, Networking, Storage and Analysis (SC’13)*. ACM, 2013, p. 55.
- [14] L. A. Barroso and U. Hölzle, “The case for energy-proportional computing,” *IEEE computer*, vol. 40, no. 12, pp. 33–37, 2007.
- [15] J. Li, K. Shuang, S. Su, Q. Huang, P. Xu, X. Cheng, and J. Wang, “Reducing operational costs through consolidation with resource prediction in the cloud,” in *Proc. of CCGRID’12*, 2012.
- [16] M. Lin, A. Wierman, L. L. H. Andrew, and E. Thereska, “Dynamic right-sizing for power-proportional data centers,” in *INFOCOM’11*, 2011.

REFERENCES

- [17] L. Wei, B. He, and C. H. Foh, “Towards Multi-Resource physical machine provisioning for IaaS clouds,” in *IEEE ICC 2014 - Selected Areas in Communications Symposium (ICC’14 SAC)*, 2014.
- [18] Z. Xiao, W. Song, and Q. Chen, “Dynamic resource allocation using virtual machines for cloud computing environment,” *IEEE Transactions on Parallel and Distributed Systems*, 2013.
- [19] M. Mao and M. Humphrey, “Auto-scaling to minimize cost and meet application deadlines in cloud workflows,” in *Proc. of SC’11*, 2011.
- [20] T. J. Hacker and K. Mahadik, “Flexible resource allocation for reliable virtual cluster computing systems,” in *Proc. of SC’11*, 2011.
- [21] G. Chen, W. He, J. Liu, S. Nath, L. Rigas, L. Xiao, and F. Zhao, “Energy-aware server provisioning and load dispatching for connection-intensive internet services.” in *Proc. of NSDI’08*, 2008.
- [22] C. Delimitrou and C. Kozyrakis, “Qos-aware scheduling in heterogeneous datacenters with paragon,” *ACM Transactions on Computer Systems (TOCS)*, vol. 31, no. 4, p. 12, 2013.
- [23] W. Wang, B. Li, and B. Liang, “Dominant resource fairness in cloud computing systems with heterogeneous servers,” in *INFOCOM’14*, 2014.
- [24] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, and I. Stoica, “Dominant resource fairness: fair allocation of multiple resource types,” in *USENIX NSDI*, 2011.
- [25] A. Ghodsi, M. Zaharia, S. Shenker, and I. Stoica, “Choosy: max-min fair sharing for datacenter jobs with constraints,” in *Proc. of the 8th ACM European Conference on Computer Systems*. ACM, 2013, pp. 365–378.
- [26] C. Joe-Wong, S. Sen, T. Lan, and M. Chiang, “Multi-resource allocation: Fairness-efficiency tradeoffs in a unifying framework,” in *Proc. of INFOCOM’12*. IEEE, 2012, pp. 1206–1214.

REFERENCES

- [27] A. A. Bhattacharya, D. Culler, E. Friedman, A. Ghodsi, S. Shenker, and I. Stoica, “Hierarchical scheduling for diverse datacenter workloads,” in *Proceedings of the 4th annual Symposium on Cloud Computing*. ACM, 2013, p. 4.
- [28] Q. Zhang, M. F. Zhani, R. Boutaba, and J. L. Hellerstein, “Harmony: dynamic heterogeneity-aware resource provisioning in the cloud,” in *Proc. of 2013 IEEE 33rd International Conference on Distributed Computing Systems (ICDCS’13)*. IEEE, 2013, pp. 510–519.
- [29] Q. Zhang, M. Zhani, R. Boutaba, and J. Hellerstein, “Dynamic heterogeneity-aware resource provisioning in the cloud,” *IEEE Transactions on Cloud Computing*.
- [30] S. K. Garg, A. N. Toosi, S. K. Gopalaiyengar, and R. Buyya, “Sla-based virtual machine management for heterogeneous workloads in a cloud datacenter,” *Journal of Network and Computer Applications*, 2014.
- [31] M. Kim, Y. Cui, S. Han, and H. Lee, “Towards efficient design and implementation of a hadoop-based distributed video transcoding system in cloud computing environment,” *International Journal of Multimedia and Ubiquitous Engineering*, 2013.
- [32] A. Ashraf, F. Jokhio, T. Deneke, S. Lafond, I. Porres, and J. Lilius, “Stream-based admission control and scheduling for video transcoding in cloud computing,” in *Proc. of International Symposium on CCGrid’13*. IEEE, 2013.
- [33] A. Ashraf, “Cost-efficient virtual machine provisioning for multi-tier web applications and video transcoding,” in *Proc. of International Symposium on CCGrid’13*. IEEE, 2013.
- [34] Z. Zhuang and C. Guo, “Building cloud-ready video transcoding system for content delivery networks (cdns),” in *Proc. of Global Communications Conference (GLOBECOM’12)*. IEEE, 2012.
- [35] R. Pereira, M. Azambuja, K. Breitman, and M. Endler, “An architecture for distributed high performance video processing in the cloud,” in *Proc. of CLOUD’10*. IEEE, 2010.

REFERENCES

- [36] A. Garcia, H. Kalva, and B. Furht, “A study of transcoding on cloud environments for video content delivery,” in *Proc. of the 2010 ACM multimedia workshop on Mobile cloud media computing*. ACM, 2010.
- [37] A. GARcIA and H. Kalva, “Cloud transcoding for mobile video content delivery,” in *Proceedings of the IEEE International Conference on Consumer Electronics (ICCE)*, 2011.
- [38] R. Cheng, W. Wu, Y. Lou, and Y. Chen, “A cloud-based transcoding framework for real-time mobile video conferencing system,” in *International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud’14)*. IEEE, 2014.
- [39] F. Jokhio, A. Ashraf, S. Lafond, I. Porres, and J. Lilius, “Prediction-based dynamic resource allocation for video transcoding in cloud computing,” in *2013 21st Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*. IEEE, 2013.
- [40] F. Jokhio, A. Ashraf, S. Lafond, and J. Lilius, “A computation and storage trade-off strategy for cost-efficient video transcoding in the cloud,” in *2013 39th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE, 2013.
- [41] W. Zhang, Y. Wen, J. Cai, and D. Wu, “Towards transcoding as a service in multimedia cloud: Energy-efficient job dispatching algorithm,” *IEEE Transactions on Vehicular Technology*, 2014.
- [42] H. Herodotou and S. Babu, “Profiling, what-if analysis, and cost-based optimization of mapreduce programs,” *Proceedings of the VLDB Endowment*, 2011.
- [43] H. Herodotou, F. Dong, and S. Babu, “No one (cluster) size fits all: automatic cluster sizing for data-intensive analytics,” in *Proceedings of the 2nd ACM Symposium on Cloud Computing*. ACM, 2011.
- [44] H. Herodotou, H. Lim, and L. etal., “Starfish: A self-tuning system for big data analytics.” in *CIDR*, 2011.

REFERENCES

- [45] V. Venkataramani, Z. Amsden, N. Bronson, G. Cabrera III, P. Chakka, P. Dimov, H. Ding, J. Ferris, A. Giardullo, J. Hoon, S. Kulkarni, N. Lawrence, M. Marchukov, D. Petrov, and L. Puzar, “Tao: how facebook serves the social graph,” in *Proc. of SIGMOD’12*, 2012.
- [46] C. Ji, Y. Li, W. Qiu, U. Awada, and K. Li, “Big data processing in cloud computing environments,” in *Proc. of International Symposium on Pervasive Systems, Algorithms and Networks*. IEEE Computer Society, 2012.
- [47] G. Jung, N. Gnanasambandam, and T. Mukherjee, “Synchronous parallel processing of big-data analytics services to optimize performance in federated clouds,” in *Proc. of CLOUD’12*. IEEE, 2012.
- [48] Z. Zhu, S. Li, and X. Chen, “Design qos-aware multi-path provisioning strategies for efficient cloud-assisted svc video streaming to heterogeneous clients,” *IEEE transactions on multimedia*, 2013.
- [49] Human Genome Project, <http://www.ornl.gov/hgmis/home.shtml>.
- [50] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica *et al.*, “A view of cloud computing,” *Communications of the ACM*, 2010.
- [51] K. Lim, J. Chang, T. Mudge, P. Ranganathan, S. K. Reinhardt, and T. F. Wenisch, “Disaggregated memory for expansion and sharing in blade servers,” in *ACM SIGARCH Computer Architecture News*. ACM, 2009.
- [52] J. Ousterhout, P. Agrawal, and e. a. Erickson, “The case for ramclouds: scalable high-performance storage entirely in dram,” *ACM SIGOPS Operating Systems Review*, 2010.
- [53] B. Debnath, S. Sengupta, and J. Li, “Flashstore: high throughput persistent key-value store,” *Proc. of the VLDB Endowment*, 2010.
- [54] R. Fang, H.-I. Hsiao, B. He, C. Mohan, and Y. Wang, “High performance database logging using storage class memory,” in *Proc. of IEEE 27th International Conference on Data Engineering (ICDE’11)*, 2011.

- [55] Spark, <http://spark.apache.org/>.
- [56] Y. Chen, A. Das, W. Qin, A. Sivasubramaniam, Q. Wang, and N. Gautam, “Managing server energy and operational costs in hosting centers,” in *Proc. of SIGMETRICS’05*, 2005.
- [57] X. Wen, K. Chen, Y. Chen, Y. Liu, Y. Xia, and C. Hu, “Virtualknotter: Online virtual machine shuffling for congestion resolving in virtualized datacenter,” in *Proc. of ICDCS’12*, 2012.
- [58] N. M. Amato, L. Rauchweger *et al.*, “Processing big data graphs on memory-restricted systems,” in *Proc. of international conference on Parallel architectures and compilation*. ACM, 2014.
- [59] M. Alrokayan, A. Vahid Dastjerdi, and R. Buyya, “Sla-aware provisioning and scheduling of cloud resources for big data analytics,” in *2014 IEEE International Conference on Cloud Computing in Emerging Markets (CCEM)*. IEEE, 2014.
- [60] U. P. Banditwattanawong T., Masdisornchote M., “Economical and efficient big data sharing with i-cloud,” in *2014 International Conference on Big Data and Smart Computing*. IEEE, 2014.
- [61] V. S. Prakash, Y. Wen, and W. Shi, “Tape cloud: Scalable and cost efficient big data infrastructure for cloud computing,” in *IEEE Sixth International Conference on Cloud Computing*. IEEE, 2013.
- [62] Y. Yuan, H. Wang, D. Wang, and J. Liu, “On interference-aware provisioning for cloud-based big data processing,” in *IEEE/ACM 21st International Symposium on Quality of Service (IWQoS)*. IEEE, 2013.
- [63] M. M. Hassan, B. Song, M. S. Hossain, and A. Alamri, “Qos-aware resource provisioning for big data processing in cloud computing environment,” in *2014 International Conference on Computational Science and Computational Intelligence (CSCI)*. IEEE, 2014.

REFERENCES

- [64] G. Ananthanarayanan, A. Ghodsi, A. Wang, D. Borthakur, S. Kandula, S. Shenker, and I. Stoica, “Pacman: coordinated memory caching for parallel jobs,” in *Proc. of NSDI’12*, 2012.
- [65] B. Fitzpatrick, “Distributed caching with memcached,” *Linux journal*, 2004.
- [66] B. Urgaonkar, P. Shenoy, and T. Roscoe, “Resource overbooking and application profiling in shared hosting platforms,” *ACM SIGOPS Operating Systems Review*, 2002.
- [67] D. Williams, H. Jamjoom, Y.-H. Liu, and H. Weatherspoon, “Overdriver: Handling memory overload in an oversubscribed cloud,” in *ACM SIGPLAN Notices*, 2011.
- [68] L. Wang, R. A. Hosn, and C. Tang, “Remediating overload in over-subscribed computing environments,” in *Proc. of IEEE 5th International Conference on Cloud Computing (CLOUD’12)*, 2012.
- [69] H. Liu, C.-Z. Xu, H. Jin, J. Gong, and X. Liao, “Performance and energy modeling for live migration of virtual machines,” in *Proc. of HPDC’11*, 2011.
- [70] X. Zhang, Z.-Y. Shae, S. Zheng, and H. Jamjoom, “Virtual machine migration in an over-committed cloud,” in *Proc. of 2012 IEEE Network Operations and Management Symposium (NOMS’12)*, 2012.
- [71] Rackspace Cloud, <http://www.rackspace.com/>.
- [72] S. Di, D. Kondo, and W. Cirne, “Host load prediction in a google compute cloud with a bayesian model,” in *Proc. of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC’12)*. IEEE Computer Society Press, 2012.
- [73] R. L. Henderson, “Job scheduling under the portable batch system,” in *Proc. of JSSPP’95*, 1995.
- [74] D. Nurmi, A. Mandal, J. Brevik, C. Koelbel, R. Wolski, and K. Kennedy, “Evaluation of a workflow scheduler using integrated performance modelling and batch queue wait time prediction,” in *Proc. of SC’06*, 2006.

REFERENCES

- [75] E. Thereska, A. Donnelly, and D. Narayanan, “Sierra: a power-proportional, distributed storage system,” *Microsoft Research Ltd., Tech. Rep. MSR-TR-2009-153*, 2009.
- [76] E. Michon, J. Gossa, S. Genaud *et al.*, “Free elasticity and free cpu power for scientific workloads on iaas clouds.” in *ICPADS*. Citeseer, 2012, pp. 85–92.
- [77] Rackspace Cloud Pricing, <http://www.rackspace.com/cloud/servers>.
- [78] Google Inc, <http://code.google.com/p/googleclusterdata/>.
- [79] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, and M. A. Kozuch, “Heterogeneity and dynamicity of clouds at scale: Google trace analysis,” in *Proceedings of the Third ACM Symposium on Cloud Computing*. ACM, 2012.
- [80] P. Dhawalia, S. Kailasam, and D. Janakiram, “Chisel: A resource savvy approach for handling skew in mapreduce applications,” in *Proc. of CLOUD’13*. IEEE, 2013.
- [81] “Cisco visual networking index: Global mobile data traffic forecast update, 2012–2017,” *Cisco Public Information*, 2013.
- [82] A. Vetro, J. Cai, and C. W. Chen, “Rate-reduction transcoding design for wireless video streaming,” *Wireless Communications and Mobile Computing*, 2002.
- [83] A. Vetro, C. Christopoulos, and H. Sun, “Video transcoding architectures and techniques: an overview,” *Signal Processing Magazine, IEEE*, 2003.