Dynamic Resource Scheduling in Cloud Data Center

Dissertation

zur Erlangung des Doktorgrades
Dr. rer. nat.
der Mathematisch-Naturwissenschaftlichen Fakultäten
der Georg-August-Universität zu Göttingen

im PhD Programme in Computer Science (PCS)
der Georg-August University School of Science (GAUSS)

vorgelegt von

Yuan Zhang
aus Hebei, China

Göttingen
im August 2015

Betreuungsausschuss:

Prof. Dr. Xiaoming Fu,
Georg-August-Universität Göttingen

Prof. Dr. Dieter Hogrefe,
Georg-August-Universität Göttingen

Prüfungskommission:

Referent:

Prof. Dr. Xiaoming Fu,
Georg-August-Universität Göttingen

Korreferenten:
der Prüfungskommission:

Prof. Dr. K.K. Ramakrishnan,
University of California, Riverside, USA

Weitere Mitglieder

Prof. Dr. Dieter Hogrefe,
Georg-August-Universität Göttingen
Prof. Dr. Carsten Damm,
Georg-August-Universität Göttingen
Prof. Dr. Winfried Kurth
Georg-August-Universität Göttingen
Prof. Dr. Burkhard Morgenstern
Georg-August-Universität Göttingen

Tag der mündlichen Prüfung: 14. September 2015

Abstract

Cloud infrastructure provides a wide range of resources and services to companies and organizations, such as computation, storage, database, platforms, etc. These resources and services are used to power up and scale out tenants' workloads and meet their specified service level agreements (SLA). With the various kinds and characteristics of its workloads, an important problem for cloud provider is how to allocate it resource among the requests. An efficient resource scheduling scheme should be able to benefit both the cloud provider and also the cloud users. For the cloud provider, the goal of the scheduling algorithm is to improve the throughput and the job completion rate of the cloud data center under the stress condition or to use less physical machines to support all incoming jobs under the overprovisioning condition. For the cloud users, the goal of scheduling algorithm is to guarantee the SLAs and satisfy other job specified requirements. Furthermore, since in a cloud data center, jobs would arrive and leave very frequently, hence, it is critical to make the scheduling decision within a reasonable time.

To improve the efficiency of the cloud provider, the scheduling algorithm needs to jointly reduce the inter-VM and intra-VM fragments, which means to consider the scheduling problem with regard to both the cloud provider and the users. This thesis address the cloud scheduling problem from both the cloud provider and the user side. Cloud data centers typically require tenants to specify the resource demands for the virtual machines (VMs) they create using a set of pre-defined, fixed configurations, to ease the resource allocation problem. However, this approach could lead to low resource utilization of cloud data centers as tenants are obligated to conservatively predict the maximum resource demand of their applications. In addition to that, users are at an inferior position of estimating the VM demands without knowing the multiplexing techniques of the cloud provider. Cloud provider, on the other hand, has a better knowledge at selecting the VM sets for the submitted applications. The scheduling problem is even severe for the mobile user who wants to use the cloud infrastructure to extend his/her computation and battery capacity, where the response and scheduling time is tight and the transmission channel between mobile users and cloudlet is highly variable.

This thesis investigates into the resource scheduling problem for both wired and mobile users in the cloud environment. The proposed resource allocation problem is studied in the methodology of problem modeling, trace analysis, algorithm design and simulation approach. The first aspect this thesis addresses is the VM scheduling problem. Instead of the static VM scheduling, this thesis proposes a finer-grained dynamic resource allocation and scheduling algorithm that can substantially improve the utilization of the data center

resources by increasing the number of jobs accommodated and correspondingly, the cloud data center provider's revenue. The second problem this thesis addresses is joint VM set selection and scheduling problem. The basic idea is that there may exist multiple VM sets that can support an application's resource demand, and by elaborately select an appropriate VM set, the utilization of the data center can be improved without violating the application's SLA. The third problem addressed by the thesis is the mobile cloud resource scheduling problem, where the key issue is to find the most energy and time efficient way of allocating components of the target application given the current network condition and cloud resource usage status.

The main contribution of this thesis are the followings. For the dynamic real-time scheduling problem, a constraint programming solution is proposed to schedule the long jobs, and simple heuristics are used to quickly, yet quite accurately schedule the short jobs. Trace-driven simulations shows that the overall revenue for the cloud provider can be improved by 30% over the traditional static VM resource allocation based on the coarse granularity specifications. For the joint VM selection and scheduling problem, this thesis proposes an optimal online VM set selection scheme that satisfies the user resource demand and minimizes the number of activated physical machines. Trace driven simulation shows around 18% improvement of the overall utility of the provider compared to Bazaar-I approach and more than 25% compared to best-fit and first-fit. For the mobile cloud scheduling problem, a reservation-based joint code partition and resource scheduling algorithm is proposed by conservatively estimating the minimal resource demand and a polynomial time code partition algorithm is proposed to obtain the corresponding partition.

Acknowledgements

I would need to thank many people. They helped me a lot during my PhD process. Without their help, this thesis could never have been possible.

I give my deepest grateful to my advisor Prof. Dr. Xiaoming Fu. It was his constant guidance, support, and encouragement that inspired me to pursue my research. Prof. Dr. Xiaoming Fu's valuable ideas, suggestions, comments and feedbacks formed a substantial input source of my work.

I also need to thank Prof. Dr. K.K. Ramakrishnan. Prof. Dr. K.K. Ramakrishnan spends many hours in discussing my problem and revising my work. His experience, vision and wisdom guide this work onto the right track.

I would thank the China Scholarship Council who supports my study and living in Germany. Without their generous financial aid, I would never have the chance to pursue my PhD. I would also thank my previous advisor Prof. Dr. Yongfeng Huang from Tsinghua University, who helps a lot in my scholarship application and job hunting.

I owe a lot of thanks to my former and current colleagues at the Computer Networks Group at the University of Göttingen, especially Dr. Jiachen Chen, Dr. Lei Jiao, Dr. Konglin Zhu, Dr. David Koll, Narisu Tao, Hong Huang, Lingjun Pu, Jie Li, Dr. Stephan Sigg and Dr. Xu Chen. They helped me during the last 4 years by discussing the work with me and facing some technical troubles together. They also act as good examples for me to learn from.

I would appreciate a lot that Prof. Dr. Dieter Hogrefe agreed to be a member of my thesis committee; I also thank Prof. Dr. Dieter Hogrefe, Prof. Dr. Winfried Kurth, Prof. Dr. Carsten Damm, Prof. Dr. Burkhard Morgenstern, and Prof. Dr. K.K. Ramakrishnan for serving as the exam board for my thesis. Their comments made this thesis more test proof.

Last but not the least, I would thank my my family. Their love, support and understanding are the best encouragement to me. To them I will dedicate this thesis.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Cloud computing provides tenants the resources and services in a pay-as-you-go manner no matter when and where the requests are submitted. To support the submitted applications at a large scale, an efficient and effectively resource scheduling scheme is an important part to both cloud providers and tenants. This chapter would specify the resource scheduling problem studied in this thesis.

In this section, the cloud scheduling problems are identified. To be specific, two steps are taken in looking at the cloud scheduling problem. In the first step, this work considers the intra-cloud scheduling, where the cloud provider aims to improve the inter-VM fragments via elaborately designing a problem-specific scheduling algorithm. In the second step, this work extends the vision into a world where user job resource demand is leveraged to further jointly reduce the intra-VM and inter-VM fragments.

1.1 Problem and Definitions: Cloud Resource Scheduling

The development of cloud computing offers the opportunity to deploy and use resizable, available, efficient, and scalable computation resources on demand. Users can request computation resources in terms of virtual machines (VM) provided by the cloud provider. Once the resources are allocated, the users would have the complete control of the computing resources. These resources are used to support various kinds of applications, such as online social networks, video streaming, search engine, email, and etc.

To efficiently and effectively schedule the user submitted applications over cloud data center is an important issue for application performance, system throughput, and resource utilization. The workload of the cloud data center is tremendous. As stated in [2], on each day, thousands of jobs are submitted to the public cloud and the peak rate would be as high as tens of thousands scheduling requests per second. In the meanwhile, the submitted jobs are also quite diverse in terms of resource demand, duration, throughput, latency and jitter [61, 74]. The massive size and diverse resource demand characteristics of the workloads make the scheduling process more difficult. Despite the effort by various studies to improve the physical machines's energy efficiency, the energy consumption of the current on-shelf physical machines is still not proportional to the load of the machine and the lower the utilization rate is, the more energy would be wasted. Hence, low utilization rate would result in a huge waste of data center energy. Furthermore, the revenue of the cloud provider will be lowered down since less workloads can be hosted.

Specifically, an efficient and effective cloud scheduler should satisfy the following requirements.

1. **Accurate or near accurate decision**. The scheduling decision is the key point for the cloud throughput and utilization, hence accuracy is the main measurement of the cloud scheduler. This requirement would also be referred to as the effective requirement of the cloud scheduler.

2. **Fast, online scheduling**. Since the scheduler needs to make tens of thousands of scheduling decisions per second [2], scheduling speed would hence become an important factor for online scheduling. This requirement would also be referred to as the efficiency requirement of the cloud scheduler.

3. **Compatible for heterogeneous workloads**. The physical machines in the cloud data center could have various resource capacity. The cloud workload would also have different resource demand, duration, and SLA requirements. The cloud scheduler should be able to make the fast yet accurate scheduling decisions w.r.t the job characteristics.

The computation resources in public cloud are provided in various types of VMs and various ways of using this VMs. For example, as stated on its website [46], Amazon EC2 provides General Purpose Instances like T2, M4 and M3; Compute Optimized Instances like C4 and C3; Memory Optimized Instance like R3; GPU Instance like G2; Storage Optimized Instances like I2 and D2. There are three ways of purchasing the instances (VMs), namely "On-Demand Instances, Reserved Instances and Spot Instances" [47].

To use the On-Demand Instances, users should purchase the VMs on an hourly basis. This is suitable for applications of short duration and spiky or unpredictable workloads.

To use the Reserved Instances, users should purchase the VMs with an upfront payment. They would then get a discount for the hourly fee. This is suitable for applications with long duration and steady workloads. By using Reserved Instances, it is possible for this kind of applications to save a lot of money.

The price of the Spot Instances is not constant but will fluctuate w.r.t the supply and demand relationships. To use the Spot Instances, the users should provide a maximum amount of fee they would like to pay which is their bid and is usually lower than the price of the On-Demand Instances. If the price of the Spot Instances goes lower than the user's bid, AWS would allocate his/her the type and number of VMs they specified. If the the price of the Spot Instances goes higher than the user's bid, the allocated VMs would be shut down. Spot Instances is suitable for applications with flexible start times and is only meaningful at a low price.

However, there are some problems with the current VM based resource provisioning and purchasing scheme.

On the task level, the fixed VM configuration would lower down resource utilization rate[1]. Although there are several VM types and configurations been provided, the actual number is still limited. Tenants need to choose among these limited configurations to cater to their resource demand. However, the resource could be wasted due to fixed VM configuration. Fig. 1.1 shows an example of how the fixed configuration would lower down the throughput.

In this example, there are two types of resources: CPU and memory. Let's assume there are four VM types with configurations and two tasks need to be allocated. The VM and physical machine configurations and tasks' resource demands are listed in the figure. If the tenants are able to estimate their resource demands accurately, both of the two tasks would choose VM 3 to accommodate the workload. In this case, two physical machines are needed

[1] In this thesis, the term "task" is used to represent the minimum granularity of the workload that cannot be split anymore; the term "applications" and "jobs" are used interchangeably to represent the tasks from the same tenant

to host the submitted tasks as shown in Fig. 1.1a. However, if flexible VM scheduling is allowed, only one physical machine would suffice as shown in Fig. 1.1b.

On the application/job level, the current VM-based resource provisioning is depart from the user concern and would further lower down the resource utilization rate. Current VM-based resource provisioning scheme require tenants to specify their resource demands in terms of the number and types of VMs they need. However, tenants are at an inferior position to estimate the VM demands without knowing the underlying VM multiplexing scheme. In addition to increasing the user burden, user specified VM demand would also lower down the cloud provider's profit. This can be illustrated in the following example Fig. 1.2.

In this example, there are also two types of resources: CPU and memory. The VM types with configurations and job resource demands are also listed in the figure. To accommodate the job resource demand, the user could use <1 VM1, 1 VM2> or <3 VM1, 0 VM2>. As shown in Fig. 1.2, the first VM demand would use 1 physical machines and the second scheme would need 3 physical machines. Since the physical machine configuration and its realtime usage status is only known to the cloud provider, to improve the utilization rate the cloud provider should take the responsibility of VM set selection for the tenants.

1.2 Dissertation Organization

The other chapters in this dissertation will be discussed as the following structure.

Chapter 2 provides a thorough overview of the existing cloud scheduling solutions. The problems of these solutions are discussed which lead to potential improvements. This chapter also provides information of several related fields to cloud scheduling such as workload characterization and flexible VM provision.

Chapter 3 states the underlying correlations between the three works in this thesis. The goal is to provide an overall solution to cloud scheduling problem. The work starts from the general VM scheduling problem inside a data center. It designs a problem specific scheduling algorithm to due with the stochastic bin-packing problem. Afterwards, the vision extends to a real world operation scenario with both the cloud provider and the user. The third work represents a special use case for cloud scheduling with mobile users. Due to the lack of computation power and battery life, one needs to deliberately select the piece of work to run on the cloud side.

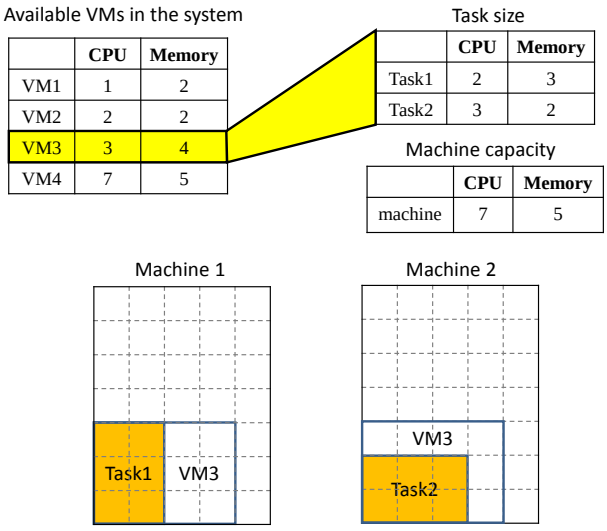
Chapter 4 studies the dynamic VM scheduling problem. Rather than using the fixed VM configuration based resource allocation, this chapter proposes a fine grained resource

scheduling scheme where each VM is shaped into the size of the user workload. A reservation-based VM scheduling scheme is proposed. Experiment results show that the flexible VM scheduling scheme could substantially improve the utilization of the data center and the job completion rate. This chapter is based-on the work in [88].

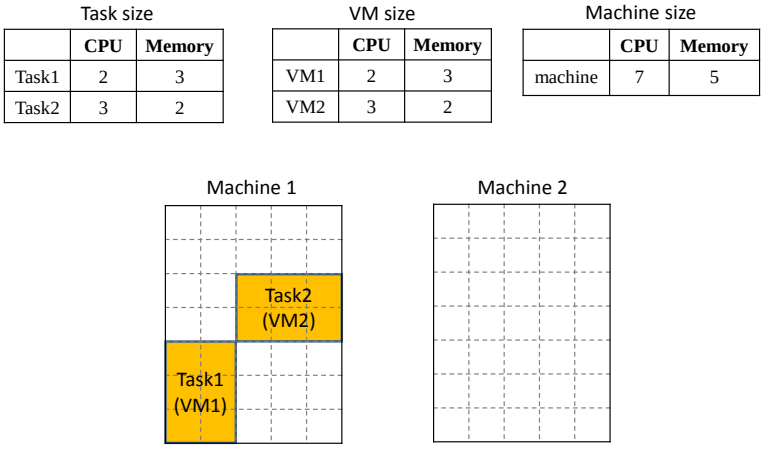
Chapter 5 studies the joint VM scheduling and VM set selection problem. This chapter proposes a multi-resource scheduler that first translate the tenants' resource demand i.,,R5ZXs; it then uses the reservation-based scheduling to allocate the VM sets onto physical machines with the goal to achieve user SLA as well as to improve the overall utility. An optimal online VM set selection algorithm is designed to satisfy the user resource demand and reduce the number of activated physical machines.

Chapter 6 studies a use case of the cloud resource scheduling problem. This work extends the initial cloud resource scheduling problem to a mobile case, where the mobile users try to leverage the remote resource to execute its computational work. The system delays include execution time on both the client and server, the transmission between the parts, as well as the code partition time on the server side. An optimal code partition algorithm is proposed and evaluations are done in real life environment. This chapter is based-on the work in [89].

Chapter 7 summarizes this work with a discussion of dissertation impact and future directions.



(a) Fixed VM configuratin.



(b) Dynamic VM configuratin.

Figure 1.1: Illustration of how fixed VM will lower down the resource utilization rate on task level.

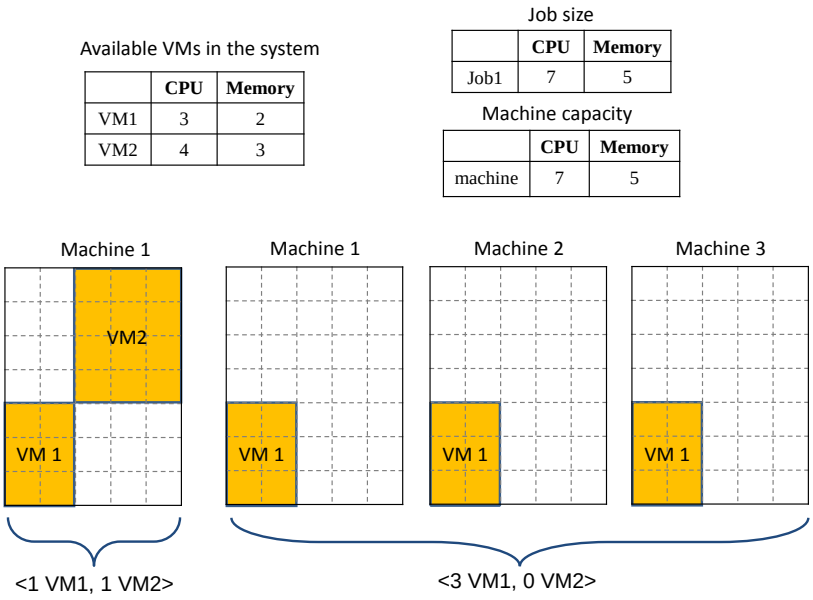


Figure 1.2: Illustration of how fixed VM will lower down the resource utilization rate on job/application level.

Chapter 2

Background and Related Work

This chapter provides an overview of the state-of-the-art work of cloud scheduling and workload characterization. It also presents the background knowledge and some terminologies of cloud resource scheduling.

2.1 Cloud Scheduling

The scale and impact of cloud data centers grow significantly during last decades. Large data centers such as Amazon EC2 and Microsoft Azure, usually contain tens of thousands of physical machines with a sophisticated network topology. In the meanwhile, the workload in cloud data center is heterogenous since it comes from various types and functions of applications. In a complicated system like this, it is usually difficult to develop scalable scheduling schemes to handle the enormous number and quite diverse jobs/tasks [60]. To be specific, it is hard for the cloud providers to make online decisions of which tasks should be run on which physical machine at what time.

A number of solutions have been proposed for the scheduling problem in large scale cloud data centers [30] [19]. The ASRPT [60] approach can provide a scheduler with an accuracy bound of two compared to the optimal solution. However, the scalability of ASPRT is not good enough since it contains a sorting procedure whose time complexity grows significantly w.r.t. the number of physical machines.

There are also some works that leverages the BvN [12] heuristic to design their scheduling policies. To use the BvN heuristic, one need to know the distribution of the arrival process. The scheduler could guarantee the queue stability by requiring an maximum waiting time for each schedule in the BvN decomposition matrix. The problem with the BvN-based schedulers is the maximum waiting time can be quite large and would grow significantly with the number of ToR switches [11]. Hence the scalability of the BvN-based scheduler is not good enough either.

There are also some other works that take the cloud pricing into their consideration. The goal is to maximize the provider profit [91] [90] or to improve the overall social welfare [92]. The problem with this kind of works is that their auction policies are derived from empirical studies of spot instances which may not reflect the future interests of the cloud provider and user.

2.2 Cloud Workload: Characterization and Prediction

A number of works have studied the public cloud workload characteristics [14]. [15] compares Amazon EC2 to several high-performance computing (HPC) clusters. [16] conducts a similar study that compares EC2 to NASA HPC cluster. Their finding is that the network utilization in the public cloud is not as high as in operational clusters. [17] deploys several networking benchmarks on Amazon EC2 and also finds that EC2 VMs has a lower resource utilization ratio and higher variability than private scientific computation cloud. [18] finds

that heterogeneity of the physical machine hardware is the key factor that increases the performance variance. [13] provides a measurement study of resource usage patterns in EC2 and Azure and finds out that a daily usage pattern.

Chapter 3

Framework and Architecture

This chapter discusses the underlying correlations between the three works. It first considers the problem from the provider's point of view with the goal to maximize the profit. The work does not stop there. It extends the problem to take the user purchase process into consideration. Interestingly, with user level information, both the cloud provider and the end user would benefit. The third part of this dissertation takes the step further onto the specific mobile users. The system architecture is shown as follows.

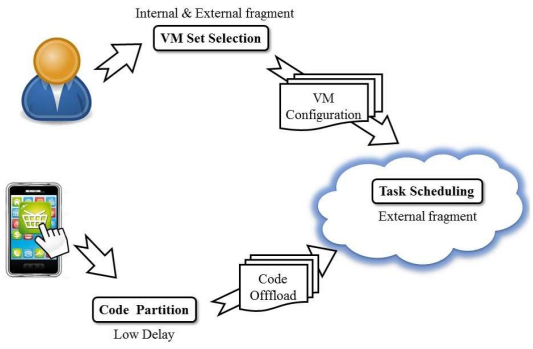


Figure 3.1: Two representative utility functions for best-effort applications.

3.1 Trace-driven VM Scheduling Algorithm

The first setting considers a scenario where the workload comes as individual tasks. The aim of this work is to improve the cloud utilization by reducing the inter-VM fragments, which we called the external fragments.

The work starts from going through literature work when three different existing scheduling approaches are studied. The problem with these work is that they cannot balance between the scheduling accuracy and scheduling speed. Hence, this work try to provide high cloud utilization within a reasonable scheduling time.

To better understand the problem specific scheduling input, a trace-driven analysis is done to study the workload characteristic. The finding is that duration is the deterministic factor of task importance.

Therefore, this work proposes a combined constraint programming and heuristic algorithm. The constraint programming algorithm is used to provide an accurate scheduling result for the long and important tasks. The heuristic algorithm is used to schedule the short and less important tasks.

A trace-driven simulation is done to evaluate the scheduling algorithm. It is worth noticing that the solution is not restrict to the two heuristic been selected. Results show that the combined algorithm can achieve around 18% CPU and memory utilization improvement.

3.2 Joint User and Cloud Scheduling

The second setting extends the vision to a real world cloud operation scenario where users have to buy VM sets from the cloud provider. There exist multiple VM sets that can satisfy the user resource demand while not all of them would have the same quality. By delicately choosing the VM set for the users, the cloud provider would further improve its resource utilization.

This part of work first illustrates why the low quality VM set would cause both internal and external fragments. Then it provides a solution to find the optimal VM set when the cloud users arrive as a stochastic process. It has be proved by theoretical analysis that the proposed algorithm can achieve global optimal.

3.3 Cloud Scheduling for Mobile Users

The third setting discussed the mobile cloud scenario when the cloud users has more restrict requirement for the timely response and local side energy saving. The work uses two applications namely the face recognition and natural language processing as examples to illustrate the joint cloud partition and resource scheduling problem.

This work proposes a scheme for multiple cloud users to share the common cloud infrastructure. It contains a mobile to cloud code partition process and cloud resource scheduling process. Since the network condition will impact the state transmission between the mobile user and the cloud provider, Instead of making an offloading decision on each node, the code partition process transfers the call graph into a call link to get the optimal offloading and integrating points.

Chapter 4

Deadline Aware Multi-Resource Scheduling in Cloud Data Center

This chapter considers the dynamic cloud resource scheduling problem on the task level. Cloud data centers typically require tenants to specify the resource demands for the virtual machines (VMs) they create using a set of pre-defined, fixed configurations, to ease the resource allocation problem. Unfortunately, this leads to low resource utilization of cloud data center as tenants are obligated to conservatively predict the maximum resource demand of their applications.

The work in this chapter argues that instead of using a static VM resource allocation, the finer-grained dynamic resource allocation and scheduling scheme can substantially improve the utilization of the data center resources by increasing the number of tasks accommodated and correspondingly, the cloud data center provider's revenue.

The dynamic real-time scheduling of tasks can also ensure that the performance goals for the tenant VMs are achieved. Examining a typical publicly available cluster data center trace, an observation is that the cloud workloads follows the 80/20 principle. A large number of tasks are short and require and only a small proportion of tasks are long and which require substantial compute or memory resources. This observation can be used to facilitate the resource scheduling algorithm design.

This work proposes an optimization based approach that exploits this division between the short and long jobs to dynamically allocate a cloud data center's resources to achieve significantly better utilization by increasing the number of tasks accommodated by the data center.

The rest of the chapter is organized as follows. Section 4.1 gives an overview of the scheduling algorithms and resource allocation strategies. Section 4.2 presents the integer programming model to describe the scheduling problem. Section 4.3 provides our com-

bined constraint programming and heuristic solution. Simulation results and evaluations are presented in Section 4.4. This chapter is summarized in Section 4.5.

4.1 Background and Motivation

Public cloud services are known for their flexibility to offer compute capability to enterprises and private users on a pay-as-you-go basis. It provides a variety of resources, such as CPU, memory, storage and network resource to tenants. An essential aspect of cloud computing is its potential to statistically multiplex user demands into a common shared physical infrastructure to improve efficiency. However, the average utilization of each active physical machine in cloud data center is still relatively low. In an ideal case, cloud services could allow the tenants to dynamically request their computing resources just based on their instantaneous needs. However, this is not quite the case in reality. According to recent reports, the estimated utilization ratio of AWS data center is approximately 7% [7], while the utilization rate at a corporate datacenter is approximately 40% -60% [42]. Despite the effort by various studies to improve the energy efficiency of physical machine, the energy consumption of current on-shelf physical machines is still not proportional to its load, and the lower the utilization rate is, the more energy would be wasted. Hence, low utilization rate would result in a waste of datacenter energy. Furthermore, since less workloads can be hosted in the system, the revenue of the cloud provider will be lowered down.

The efficiency of a cloud infrastructure relies heavily on the underlying resource allocation mechanism. The de-facto policy adopted by today's cloud operators is virtual machine (VM)- based resource allocation, where a cloud operator provides a set of pre-defined, fixed VM configurations. To get access to the resource, tenants need to choose among these limited configurations and specify the time and number of VMs they need. The cloud operator then allocates the corresponding resources to the tenants. One issue of this approach is that users have to purchase the resources based on their estimation of each VM's maximum resource demand. However, the average consumption is often lower than the peak value, resulting in resource wastage. Recently, several proposals have been made for resource scheduling of cloud infrastructures with an aim to improve the resource efficiency [21, 61, 90]. However, the inherent problem of resource waste due to fixed VM configuration limits the gains.

This work argues that using finer-grained resource scheduling can benefit both the cloud operator and the tenants. In addition to evaluation metrics of job completion rate and resource utilization rate, we also consider the overall revenue of the cloud operator as the optimization objective which can be seen as a direct reflection of the cloud operator's interest. This also results in meeting the job's deadline. With objective of maximizing the revenue and taking into account the data center's capacity constraint, the multi-resource scheduling problem is modeled as an integer programming problem. However, the traditional integer programming solvers are not applicable here due to the strict requirement for online resource scheduling to determine the solution quickly and schedule the job so that it

completes within the deadline. As summarized in [24], even for a very small scale of jobs and number of servers, the time to solve the problem would be unacceptable. For example, in one case tested with 4 machines and 22 jobs, the solution time is 1957.6 seconds on Intel Xeon E5420 2.50 GHz computers with 6 MB cache and 6 GB memory.

The work analyzes the resource demands of jobs from a public cluster trace [49], and find out that they have a clear relationship to the job duration. The relatively small number of long duration jobs consume the majority of the computing resources. This motivates the approach to get a tradeoff between scheduling accuracy and scheduling speed: since the few long duration jobs consume most of the resources, a sophisticated scheduling algorithm that can reach an optimal solution is used; on the other hand, for short duration jobs, due to their large number, fast scheduling is important. Therefore, a combined constraint programming and heuristic scheduling algorithm is proposed. We use constraint programming as the tool to find an optimal scheduling for the long duration jobs. For the jobs with short durations, our experiments show that simple heuristics such as first fit or best fit suffice.

4.2 System Model

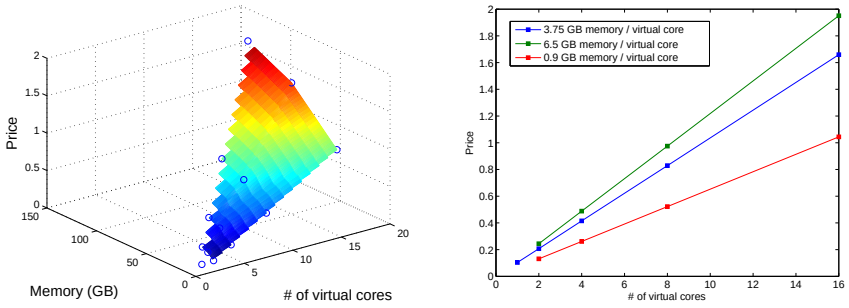
This work models the multi-resource scheduling problem in a public cloud as an integer programming problem with an objective to maximize the overall revenue of cloud operator. The overall revenue is defined to be the sum of the utility of jobs that have met their deadlines. For jobs that cannot get allocated, they do not contribute to the cloud operator's revenue.

4.2.1 Pricing Policy

Multi-resource pricing is challenging for public cloud operators. Many approaches have been proposed to get a fair, profitable, and proportional pricing scheme for the multi-resource environment [84] [26] [27] [29]. Since the pricing policy is orthogonal to the resource scheduling problem, two simple pricing policies are used in this work, as presented below.

4.2.1.1 Market Pricing

The first pricing policy been used is the policy adopted by current cloud operators [48]. Currently the Google Compute Engine provides 3 VM types, namely standard, high memory and high CPU. The configurations that are not covered by the policy are calculated by



(a) Interpolation of market pricing.

(b) Nonlinearity with memory.

Figure 4.1: Market pricing policy and its nonlinearity with memory usage.

interpolation. Fig. 4.1a shows the interpolation results. A take-away message from the market pricing policy is that the price grows nonlinearly with memory usage. As can be seen from Fig. 4.1b, the price grows linearly with the CPU usage, however the gap between memory usage is nonlinear. When the amount of memory increases, the incremental cost decreases.

4.2.1.2 Linear Pricing

For simplicity a linear pricing model is used for comparison. Based on current pricing by cloud providers, CPU plays a more important role for pricing than memory does. Hence the linearity of price is based on the CPU requirements. The baseline been used is the standard VM type in Google Compute Engine [48] where the price will increase by 0.104 if the number of virtual cores increase by 1.

4.2.2 Integer Programming

The goal of resource scheduling is to maximize the revenue for all the jobs completed with the capacity constraint on each machine. We assume that a job is fine-grained enough that it needs no more than 1 physical machine to execute. Furthermore, we do not allow job preemptions in our system. This nonpreemption requirement implies that we immediately know whether a job contributes to the overall revenue the moment it gets scheduled.

Let $\mathbf{R}_{j,t}$ denote the resource demand of job j at time t. Let's consider a multi-resource scenario here, thus $\mathbf{R}_{j,t}$ is a vector. $\mathbf{C}_m$ is the capacity of machine m, which is also a vector.

Notice that the homogeneity property is not imposed on the resource capacity of machines. $x_{m,j,t}$ is a Boolean variable, where 1 indicates job j is assigned to machine m at time t and 0 otherwise. $\mathbf{L}_{m,t}$ is the utilization vector of machine m at time t. All machines shouldn't exceed their capacities at anytime. D_j is the deadline of job j while F_j is its finishing time. U_j is the utility of job j. Let's define U_j to be the w_j which is the revenue obtained for a job j if it completes before its deadline and 0 otherwise. This leads to the following integer programming model:

$$\max \sum_{j \in J} U_j \tag{4.2.1}$$

$$s.t. \sum_{j \in J} \mathbf{R}_{j,t} \times x_{j,m,t} = \mathbf{L}_{m,t} \quad \forall m \in M, t \in T \tag{4.2.2}$$

$$\sum_{j \in J} \mathbf{L}_{m,t} \leq \mathbf{C}_m \quad \forall m \in M, t \in T \tag{4.2.3}$$

$$x_{j,m,t} \in \{0,1\} \tag{4.2.4}$$

$$\sum_{m \in M} x_{m,j,t} \in \{0,1\} \quad \forall j \in J, t \in T \tag{4.2.5}$$

$$\sum_{t \in T, t>0} |x_{j,m,t} - x_{j,m,t-1}| \leq 2 \tag{4.2.6}$$

$$U_j = \begin{cases} w_j & F_j \leq D_j \\ 0 & F_j > D_j \end{cases} \tag{4.2.7}$$

4.3 Combined Constraint Programming and Heuristic Algorithm

The combined problem of job scheduling with resource constraints and deadline requirements is NP-hard [33]. There are two challenges that make the existing approaches using an integer programming model impractical in this situation. One is that a global optimal solution needs information (arrival time, resource demand) of future jobs, which would cause the system become non causal. Another is that unlike allocation problems, a scheduling problem requires an online solution. The scale and speed requirements are difficult to achieve by current integer linear programming solvers. Thus, a new approach is needed for this multi-resource scheduling problem.

The scheme been adopted here is the reservation mechanism, a practical approach used by parallel computing to provide more reliable service to high priority jobs [38]. It fits the problem well since the portion of more privileged jobs is small and their duration is long; once they get accommodated, they usually occupy that resource for a while hence allow the scheduling engine enough time for the next iteration.

Constraint programming (CP) is a set of tools that provides a high performance solution to constraint-based discrete variable problems using constraint propagation and decision-making based search. The optimization engine of CP software is a set of logical deductive inferences rather than the relaxations techniques of integer programming algorithms. The CP solutions for resource allocation and planning have been shown to be usually 10+ times faster than the integer programming solvers for resource scheduling [56, 78].

4.3.1 Trace Analysis

The Google Cluster Trace from a 12,000-machine cluster over about a month-long period [49] is used in our study. The trace contains a trace of resource demand over time and a trace of machine availability. Resource requests and measurements are normalized to the largest capacity of the resource on any machine in the trace (which is 1.0). Due to normalization, CPU and memory usage range from 0 to 1.

Efficient VM scheduling scheme would demand a thorough understanding of the resource usage pattern of the submitted applications. The basic step of resource usage pattern analysis is to accurately yet efficiently classify the workloads into groups. To get a clear understanding of the workload features, the K-means (a multi-dimension statistical clustering algorithm) algorithm [74] is used to determine the classification of large and small in each of the 3 dimensions of a job (CPU, memory and duration).

One extream direction for job classification is to define each job as a class. However this approach would cause a large number of classes when there are many jobs which is a typical case in public cloud. On the other extream direction, is to use single class, which would provide no benefit for the resource scheduling problem. The tradeoff used here is to define two types on each of the resource dimensions.

The initial 8 classes are summarized in Table 4.1. The numbers are the mean values of all jobs within that class and the unit of duration is hours. As can be seen from the table, duration shows a clearer distinction of small and large than CPU and memory. Using duration as the clustering factor, the jobs with a duration < 2.06 hours are classified as small or short duration jobs, while those with a duration $>= 2.06$ hours are defined as large or long duration jobs.

4.3.2 Constraint Programming and Gecode Solver

In our multi-resource datacenter scheduling problem, each job has a start and end time, and a resource demand for each kind of resource. Each physical machine can execute multiple

Table 4.1: Classification in All Dimensions.

Initial classes	Duration	CPU	Memory	Percentage
SSS	0.0833	0.03	0.15	29%
SSL	0.0890	0.05	0.37	32%
SLS	0.2387	0.12	0.21	11%
SLL	0.5731	0.12	0.59	6%
LSS	19.34	0.05	0.49	17%
LSL	21.23	0.05	0.47	1%
LLS	19.57	0.11	0.12	1%
LLL	21.65	0.53	0.48	3%

jobs as long as the aggregate resource demand does not exceed the machine capacity.

Solving the scheduling problem for jobs with deadlines using integer programming, even for a single resource with constraints on the aggregate demand, can take a long time [78].The problem of accumulative resources of one resource is already considered prohibitively hard for integer programming solver. In our multi-resource datacenter scheduling problem, each job has a start and end time, and a resource demand for each kind of resource. Each physical machine can execute multiple jobs as long as the accumulated resource demand does not exceed the machine capacity. The problem of accumulative resources with deadline requirements is known to take long time for integer programming solver.

Fortunately, it matches with the "cumulatives" constraint [57] provided by the multi-resource scheduling package in the Gecode solver [50], where the resource demand of each job can be specified in the TASKS parameter set. Therefore, we chose CP to solve the datacenter multi-resource scheduling problem. The declaration of the "cumulatives" function [20] from the Gecode solver is shown as follows code 4.3.2.

Listing 4.1: Cumulatives Propagator.

```
void Gecode::cumulatives  ( Home  home,
  const IntVarArgs &  m,
  const IntVarArgs &  s,
  const IntVarArgs &  p,
  const IntVarArgs &  e,
  const IntVectorArgs &  u,
  const IntVectorArgs &  c,
  bool  at_most,
  IntConLevel  icl = ICL_DEF
 )
```

As stated in [20], in the "cumulatives" function, m denots the machine assigned to the job; s is the start time assigned the job; p is the processing time of the job; e is the end time assigned to the job; u is the amount of resources consumed by the job; c is the capacity of each machine; both u and c are vectors for each input.

Even though the CP solver is much faster than the integer linear programming solution approaches, it still cannot catch up with the pace of job arrival and departure events of just the long duration jobs. To speedup the search process, a technique is used to avoid repeatedly visiting the same search point for each iteration. This work exploits the fact that no preemption is allowed in our system. Hence, a job runs to completion on the same machine. Leveraging this property, the number of variables that need to be determined for each iteration is reduced significantly. For example, the number variables is reduced from 1,458,176 to 6,144 in one iteration.

4.3.3 Heuristic Algorithms

We use two basic 2-dimensional first fit and best fit algorithms as the heuristic for scheduling short jobs. While the corresponding 1-dimensional first fit and best fit algorithms are thoroughly studied and tested, the definition of their 2-dimensional counterparts is somewhat more flexible in its definition to match the application scenario. We define the notion of 2-dimensional first fit and best fit in our scheduling problem as follows.

4.3.3.1 2-dimensional First Fit

In the 2-dimensional first fit, we assign an incoming job to the first machine that can execute the job within its capacity. In the 3-layer tree topology for the datacenter, we define the order of assignment as being from the leftmost machine moving to the right.

4.3.3.2 2-dimensional Best Fit

In the 2-dimensional best fit heuristic, the incoming job is assigned to the machine that has a residual capacity that is closest to the job's requirements. More precisely, the heuristic is

$$\arg\min \left\{ \frac{m_r}{m_j} + \frac{c_r}{c_j} \right\} \quad m_r \geq m_j, c_r \geq c_j \tag{4.3.1}$$

4.3.4 Combined Algorithm

The combined algorithm and the heuristic algorithms are presented in Algorithm 5 and Algorithm 6, respectively. To schedule pending jobs, we adopt a basic deadline-based approach as in Algorithm 7.

Based on the statistics of job resource demands, we reserve a portion of machines for accommodating large jobs (see further discussions on best reservation ratio in Section V.E) When a large job arrives, it is scheduled (using CP) to one of the reserved machines. However, it is still possible that due to the dynamics of the job resource demand, a reserved machine is not available to execute this newly arrived job. In this case, this job will get scheduled to an unreserved machine using the heuristic algorithm.

Algorithm 1 Combined CP and Heuristic Algorithm

1: **Symbols:** α – duration threshold (2.06 of the Google trace case), H – type of heuristic (either best fit or first fit)
2: When {a job J arrives}
3: **if** $Job\_Duration < \alpha$ **then**
4: *Heuristic_scheduler(J, H)*; assign J to an unreserved machine
5: **else**
6: *CP_scheduler()*; assign a reserved machine to J
7: **if** CP failed to assign a reserved machine to J **then**
8: *Heuristic_scheduler(J, H)*; assign J to an unreserved machine
9: **end if**
10: **end if**
11: When {a job J running in machine M finishes}
12: Release machine M resources for J
13: $Schedule\_a\_queued\_job(M)$

4.4 Evaluation

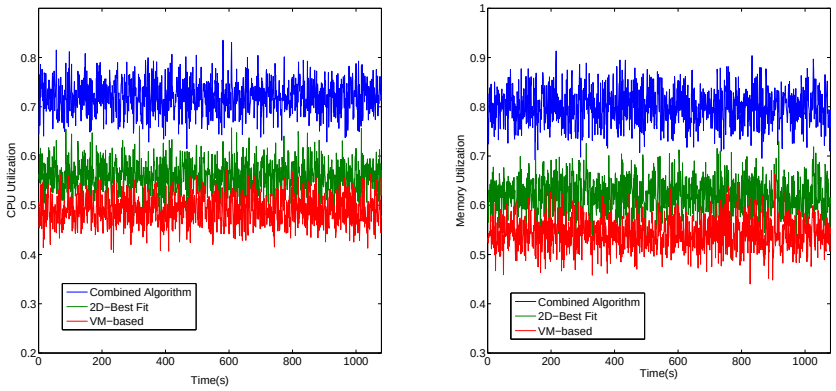
We perform trace-driven simulations to demonstrate the performance benefits of our approach. We use the public Google Cluster Trace [49] which contains 500 files of job specifications ordered by their arrival time. Each file has information for approximately 5,000 jobs. We consider a datacenter with 1,024 machines, and each physical machine is considered to have 1 unit of CPU and memory. The datacenter network is the standard 3-layer tree topology, where each of the Top-of-the-Rack (ToR) switches is connected to 16 ma-

Algorithm 2 Heuristic_scheduler(J, H)

Symbols: J – job to be scheduled, H – type of heuristic
if $H = First\_Fit$ **then**
    for Machine $M \in$ Unreserved_cluster **do**
        if $Capacity(M) \geq Demand(J)$ **then**
            Assign job J to machine M
        end if
    end for
else
    for Machine$M \in$ Unreserved_cluster **do**
        Find min $\left\{\frac{m_r}{m_j} + \frac{c_r}{c_j}\right\}$ $m_r \geq m_j, c_r \geq c_j$; 2D-best fit
    end for
    Assign job J to machine M
end if
if $No\_unreserved\_machine\_available$ **then**
    Insert job J to the deadline-ordered queue Q
end if

Algorithm 3 Schedule_a_queued_job(M)

Symbols: M – available machine
for Job $J \in$ job queue Q **do**
    if $Capacity(M) \geq Demand(J)$ **then**
        Assign job J to machine M **return**
    end if
end for



(a) CPU utilization.

(b) Memory utilization.

Figure 4.2: Resource utilization: our approach vs. heuristic approaches.

chines. Each aggregation switch is connected to 8 ToR switches and there are 8 aggregation switches that are then linked by a core switch.

4.4.1 Resource Utilization

The CPU and memory utilization for our combined algorithm using the market pricing model is shown in Fig. 4.2. The results are averaged by scheduling the jobs specified in each of the 500 files, across all the 1,024 machines. We align the time of the files to the same startup time and evaluate the variation in the utilization over time.

Fig. 4.2 shows that our combined algorithm can achieve about 20% higher CPU and memory utilization than the alternatives of a pure 'best-fit' and the VM-based scheduling policies. Thus, the combined algorithm completes a correspondingly larger number of jobs than the alternatives and more efficiently uses resources. This indicates in an oversubscription scenario, our approach allows more efficient use of resources.

4.4.2 Completion Rate

The completion rate indicates the proportion of jobs that are completed before their deadline. The average completion rate is shown in Tables 4.2 and 4.3 under the two pricing models. Note that since the datacenter as a whole is overloaded, the completion ratio for all the algorithms are not much higher than 50-60%. As can be seen from both tables, the

proposed combined algorithm can improve the overall completion rate, especially for large jobs. This result reflects the fact that large jobs can use both the reserved resources and the unreserved resources. The completion ratio for the short jobs is significantly lower, showing the effectiveness of the scheduling with reservations for the large jobs.

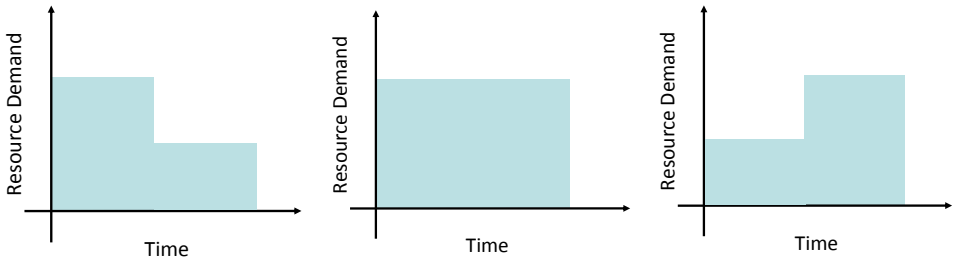
4.4.3 Comparison to Pure CP and Pure Heuristic Algorithms

Table 4.4 compares our combined algorithm at its best reservation ratio against a pure Constraint Programming algorithm or a pure heuristic (2-dimensional best fit). The overall revenue is normalized with respect to the pure CP solution for comparison purposes. The computation time for scheduling is estimated based on running the algorithm on a workstation with quad-core 2.0GHz processor and 16GB RAM. Since the CP simulation takes a long time and all traces are of a similar nature, we compare the performance on one typical trace out of the 500 traces. As can be seen from the table, although CP can achieve the best performance, its time complexity makes the approach impractical. Our combined algorithm on the other hand, can provide reasonably good results within a short amount of time for scheduling (12 minutes for our combined algorithm, 3 minutes for heuristic and 1,775 minutes or 29.6 hours for the pure CP). Our combined algorithm improves the overall revenue: going from 0.59 for the 2-dimensional best fit heuristic to 0.76 (28.8% increase). It is only 24% lower than CP. Finally, our algorithm improves the CPU utilization (28.6% higher than heuristic) and memory utilization (29% higher than heuristic).

4.4.4 Impact of Job Demand Profile

The profile of the individual job's resource demand over time also affects the efficiency of the scheduling algorithm. By far this work assumes a uniform distribution which means a job will keep consuming the same amount of resource within its duration as in Fig. 4.3b. Intuitively, the performance of the scheduling algorithm could benefit from priori information of the job characteristics. This work considers three representative profiles shown in Fig. 4.3, which are used to represent three different types of jobs: a) an initial high demand followed by a reduced demand, b) constant resource demand during the job lifetime (which is used for the evaluation so far), and c) increased demand after a period. Since the Google Trace does't have the field to specify the changes of the workloads, the change point of the first and the third shape is decided to be in the middle of its duration. The performance of our combined algorithm is shown in Table 4.5, where the overall revenue is normalized to the case when the demand is constant(profile (a)) for comparison purposes.

As can be seen from Table 4.5, the increase in computation time for scheduling large jobs



(a) Shape a: decreasing. (b) Shape b: rectangular. (c) Shape c: increasing.

Figure 4.3: Three representative job demand profiles.

Table 4.2: Completion rate (market pricing model).

Category	VM-based	2D-First	2D-Best	Combined First	Combined Best
Long jobs	-	-	-	93.7%	94.6%
Short jobs	-	-	-	43.1%	44.3%
Overall	36.7%	43.9%	45.2%	54.2%	56.1%

is acceptable (12-16 minutes, rather short considering the 2.06 hour threshold we use for the duration of large jobs). The increase of the computation time is primarily because of the recalculation of what the remaining resources are upon a job arrival. This is needed for both the CP and the heuristic components of our algorithms. The benefit of our ability to factor in the job profile, rather than thinking of the resource demand as being constant is the improvement in overall revenue: for the second and third profiles it is 14% and 16% respectively.

Table 4.3: Completion rate (linear pricing model).

Category	VM-based	2D-First	2D-Best	Combined First	Combined Best
Long jobs	-	-	-	96.7%	97.6%
Short jobs	-	-	-	41.9%	43.2%
Overall	34.0%	42.2%	44.0%	54.0%	55.2%

Table 4.4: Our combined algorithm vs. pure CP vs. 2D-best fit.

Category	Constraint Programming	2D-Best	Combined Algorithm
Computation time for scheduling	1775min	3min	12min
Overall revenue	1	0.59	0.76
CPU utilization	89%	56%	72%
Memory utilization	94%	62%	80%

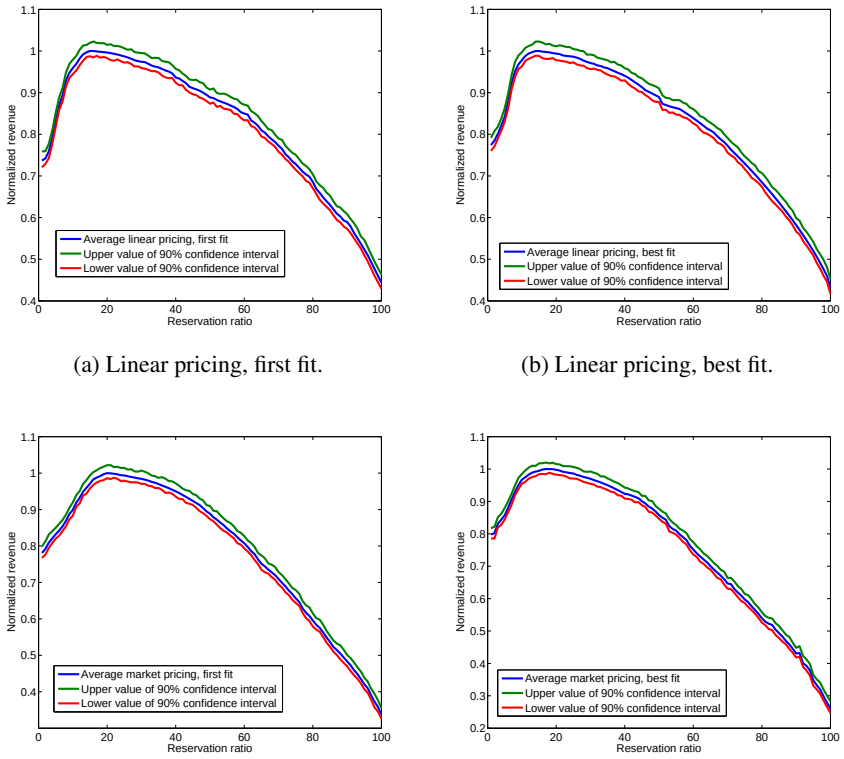
Table 4.5: Performance with 3 different representative job profiles.

	Profile (a)	Profile (b)	Profile (c)
Computation time for scheduling	12min	16min	16min
Overall revenue	1	1.14	1.16
CPU utilization	72%	82%	84%
Memory utilization	80%	89%	91%
Completion rate	56.1%	64.4%	66.1%

4.4.5 The Impact of Reservation Ratio

We evaluate the average revenue for all jobs in each of 500 files as the reservation ratio ranges from 0 to 100 percent, and the results for two pricing models are shown in Fig. 4.4. As shown in Fig. 4.4, the overall revenue increases with the reservation ratio at first, and then drops. This is consistent with our intuition that when more jobs get scheduled using the optimal solution, the overall revenue would increase as long as the resources are available for the incoming large jobs. However, when the reservation ratio gets too high, more small jobs couldn't be accommodated, hence the revenue would begin to decrease. The upper and lower bounds of the 90% confidential interval are quite close to the mean value, which means the result is quite consistent across the 500 different trace files.

When the first fit heuristic is used, the best reservation ratio is 15% using the linear pricing model compared to 20% using market pricing. The best fit case also shows a similar trend: the best reservation ratio for linear pricing is lower than that for market pricing (see Fig. 4.5 and Table 4.6). With market pricing, short jobs appear to play a more important role. Because of this, the effect of reservation may not be as obvious as in the linear pricing model. Fig. 4.5 also shows 2-dimensional best fit always slightly outperforms 2-dimensional first fit.



(a) Linear pricing, first fit.

(b) Linear pricing, best fit.

(c) Market pricing, first fit.

(d) Market pricing, best fit.

Figure 4.4: Impact of reservation ratio to the revenue of combined algorithm.

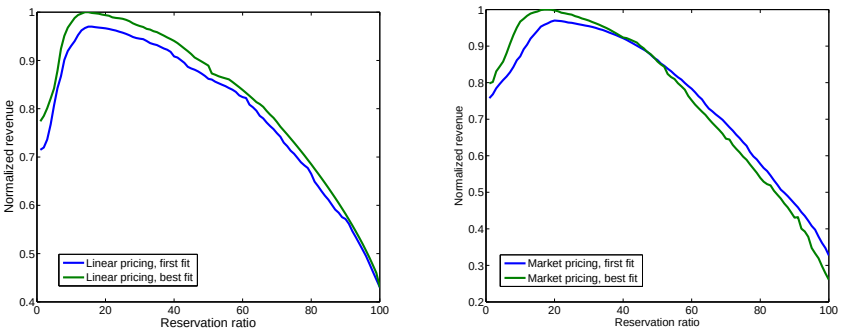
4.5 Chapter Summary

Currently, the cloud provider use fixed, pre-defined VM settings to cater to the resource demand from the tenants. However, this cause the low utilizatin rate and thus low profit of the cloud provider.

This chapter proposes a fine-grained resource scheduling approach to improve the datacenter's resource utilization. As scheduling is done in a granularity finer than VM, an online scheduling scheme is required. Leveraging the distribution pattern between short and long jobs, this chapter introduces a combined constraint programming and heuristic based scheduling algorithm to achieve fast, yet accurate scheduling. Trace-driven simulations show that this approach can improve the overall revenue and resource utilization by 25-30% over pure heuristic based scheduling.

Table 4.6: Summary: best reservation ratio in different parameters.

	Linear pricing	Market pricing
First fit	15%	20%
Best fit	14%	18%



(a) Linear pricing. (b) Market pricing.

Figure 4.5: Impact of reservation ratio under different pricing models.

Chapter 5

VM Set Selection

Current cloud resource allocation schemes usually require the tenants to specify their resource demand in terms of the number of VMs in spite of the fact that the tenants are at an inferior position to estimate the VM demands. This chapter shows that there may exist multiple VM sets that can support an application's resource demand and by elaborately select an appropriate VM set, the utilization of the data center can be improved without violating the application's SLA.

This chapter proposes a multi-resource scheduler that allocates applications to VM sets and then to physical machines with the goal to achieve user SLA and to improve the overall utility. This work contains four parts: 1. an optimal online VM set selection scheme satisfying the user resource demand and minimizing the number of activated physical machines; 2. an integer programming model to maximize the overall utility of the cloud provider under the capacity constraint; 3. a reservation-based combined constraint programming approach to the problem; 4. a trace driven simulation to evaluate our VM set selection and VM scheduling approach.

The rest of the chapter is organized as follows. Section 5.1 and 5.2 illustrates the VM set selection and scheduling problem and presents the system model respectively. Section 5.3 provides a greedy algorithm to select the VM set for each application. The proof of its optimality is in both the static and dynamic scenario. Simulation results and evaluations are presented in Section 5.4. This chapter is concluded with discussions on some open issues in Section .

Table 5.1: Application resource demand and VM & PM capacity.

	Res1	Res2
App1	15	13
App2	6	14
VM1	5	3
VM2	2	4
PM	7	7

5.1 Background and Motivation

Today's data center applications diverse in their rich variety of functions, such as video streaming [39], websites [40] and data analysis [41] etc. These applications usually demand various kinds of resources (CPU, memory, storage, etc.), on different levels and occupy the allocated resources for different time periods. The resource demand are then packed into a set of pre-defined VM configurations provided by the cloud service provider [45, 48]. For example, Amazon EC2 offers balanced resource VMs of M2 instances, compute-optimized VMs of C3 and C4 instances, memory-optimized VMs of R3 instances, storage-optimized VMs of I2 and HS1 instances and GPU-intensive VMs of G2 instances. In current cloud datacenters, users are obliged to specify their VM requirements. For example, a user may specify her VM requirement as 10 C3 VM and 2 G2 VM, each for 7 days. However, the direct concern of a cloud user is usually not the VM type and numbers, but rather the SLA requirements [71], such as completion time and resource demand. In the meanwhile, the providers would also benefit if users are allowed to just provide their resource demands. Table. 5.1 is a simple example. Let's consider a homogenous system with 2 types of resources and 2 types of VMs. The specifications of the application's resource demand and the VM and physical machine capacity is listed in 5.1.

In this example, both App1 and App2 have more than one Pareto optimal VM sets[2]. The Pareto optimal for App1 is 5 VM1, <3 VM1, VM2>, <12 VM1, 3 VM2>, <1 VM1, 5 VM2>, or 8 VM2. The other VM sets would be no better than these Pareto optimal VM sets. If use 5 VM1 or 1 VM1 and 5 VM2, then it would need 5 physical machines. If use 3 VM1 and 1 VM2 or 2 VM1 and 3 VM2, then 3 physical machines would suffice. However, 8 VM2 would use 8 physical machines. Similarly, the other applications would also have multiple VM combinations. An insight from this example is that there could be flexible VM sets to satisfy the resource demand of the application. Among the multiple possible VM sets, there could be one or more solutions that use less physical machines than the

[2]The Pareto optimal VM set means a VM combination in which it is impossible to reduce the number of VMs for any type of without increasing the number of VMs of another type.

others. By elaborately select an appropriate VM set, the utilization of the data center can be improved.

Recent studies like [58, 61, 70, 74, 79] show that the applications run on cloud have diverse service level requirements in terms of throughput, latency, and jitter. They further illustrate that the applications can be coarsely classified into two types: deadline-sensitive applications and best-effort applications [61,70,80]. The deadline-sensitive applications often consume large amount of resource and have long durations. Furthermore, they are often critical to business, and have strict deadline requirements. The best-effort applications typically consume less resource and do not have explicit deadlines. However, finishing the job at an earlier time would improve the users' satisfaction. Hence we propose a cloud scheduling system aiming at guaranteeing the deadline requirements of the deadline-sensitive applications as well as to improve the completion time of the best-effort applications. Through extensive trace-driven analysis, we find that the number and the resource consumption of the deadline-sensitive and best-effort applications follows the 80/20 principle. While majority of these applications are best-effort applications, up to 80% of the cloud data center resource is consumed by deadline-sensitive applications [61, 74, 79]. We will leverage this property to design an efficient resource scheduling algorithm later.

In this chapter, it is shown that the multiple Pareto optimal VM set property of the applications can be used to improve the overall utilization of the system. We propose a multi-resource scheduler that allocates applications to VM sets and then to physical machines with the goal to minimize the number of activated physical machines as well as to maximize the overall utility. The utility function of the deadline-sensitive applications is defined as a 1-0 function where the utility would stay the same when it is within its deadline and drop to zero when the it exceeds its deadline. For the best-effort applications, a soft utility function is used where the utility drops gradually [70]. This chapter first provides a greedy yet optimal algorithm to find the VM set that can minimize the activated physical machines and still satisfy the resource requirement. Using the VM sets as input, we model the VM scheduling problem as an integer programming problem with the goal to improve the overall utility of all applications. By leveraging the 80/20 property of the deadline-sensitive applications and best-effort applications, we propose a combined constraint programming and heuristic algorithm. We reserve a dynamic number of physical machines for the deadline-sensitive applications and use constraint programming to get a more accurate scheduling result. Due to the huge number and the less resource consumption of the best-effort applications, we use heuristic scheduling policy to get a fast yet acceptable scheduling result. Trace driven simulation shows over 25% improvement of the overall utility compared to best-fit and first-fit using the optimal VM selection algorithm, over 30% improvement compared to best-fit and first-fit using random VM selection scheme and 18% improvement compared to the Bazaar-I approach.

The main contributions of this chapter are summarized as follows:

1. according to recent studies of cloud workload heterogeneity, we classify the user applications into two types, namely deadline-sensitive applications and best-effort applications. We demonstrate that allowing application specific SLA would improve the cloud utilization and in the meanwhile guarantee the application performance.

2. we illustrate the multiple VM combination problem of application-aware scheduling and propose a greedy algorithm to choose the optimal VM combination in the sense of minimizing the activated physical machines

3. we develop a combined constrained programming scheduling algorithm exploring the 80/20 property of the two types of cloud workload which provides an accurate scheduling for deadline-sensitive applications and fast yet acceptable scheduling for best-effort applications

4. we use trace-driven simulation to evaluate our algorithm with comparison of both the VM set selection and scheduling algorithm.

5.2 VM Set Selection and Scheduling Model

This section illustrates the VM set selection and scheduling problem which aims to satisfy the user resource demand, meanwhile minimizing the number of activated physical machines. We first identify the user SLA requirements of two types of applications and then provide a model to minimize the activated physical machines while satisfying the resource demand. After that, we present the overall model aiming to improve the utility over all the applications.

5.2.1 User SLA Requirement and Utility Function

There are two aspects considered for SLA requirements of an applications:

1. *resource demand and duration*: an application use resource bundle [61], such as <16GB memory, 2 2.60GHZ CPUs, 100G SSD, 2 hours> to specify its resource demand. Both deadline-sensitive and best-effort applications should provide their resource demands and durations.

2. *deadline requirement*: each deadline-sensitive applications would provide an exact deadline associated with its resource demand. The best-effort applications do not

have such requirements.

In addition to the SLA requirements, each application has a utility function which depends on the completion time. For the deadline-sensitive applications, we use a 1-0 utility function which drops directly to zero after the deadline. As for best-effort applications, we allow different types of decreasing functions [70].

5.2.2 Minimizing Active Physical Machines

Without loosing generality, let's assume there are R types of resources and I types of VMs in the cloud system where $\{VM_1, VM_2, ..., VM_I\}$ denote the I types of VMs. The capacity of the type i VM on each type of resources is $(b_{i,1}, b_{i,2}, ..., b_{i,R})$ while the capacity of the physical machine is $(c_1, c_2, ..., c_R)$.

For each application, a feasible VM set should be able to support its resource demand on every resource dimension. Assume for application k, the resource demand for type r is $a_{k,r}$, and $x_{i,k}$ is the number of VM_i allocated to that application. Hence, the resource demand for application k would be:

$$\sum_{i=1}^{I} b_{i,r} \times x_{i,k} \geq a_{k,r} \quad \forall r \in \{1,2,...,R\}, k \in \{1,2,...,K\} \tag{5.2.1}$$

Assume we have M physical machines and K applications in total. For each physical machine, the VMs allocated to that physical machine should not exceed its capacity on every resource dimension. Let z_i be the number of type i VM allocated on a physical machine. Then the capacity constraint on that physical machine would be,

$$\sum_{i=1}^{I} b_{i,r} \times z_i \leq c_r \quad \forall r \in \{1,2,...,R\} \tag{5.2.2}$$

Actually, the capacity of a physical machine can be calculated in a simple way. In a system of I types of VMs, the maximum number of VM1 that can be accommodated on a physical machine would be $\min\{\frac{c_1}{b_{1,1}}, \frac{c_2}{b_{1,2}}, ... \frac{c_R}{b_{1,R}}\}$, which is the case when only VM1 and no other VMs are allocated. Similarly, the maximum number of VM2 that can be accommodated on a physical machine would be $\min\{\frac{c_1}{b_{2,1}}, \frac{c_2}{b_{2,2}}, ... \frac{c_R}{b_{2,R}}\}$. More generally, the maximum number of VM_i would be $\min_{i \leq I, r \leq R}\{\frac{c_i}{b_{i,r}}\}$. Let's denote v_i to be the maximum number of VM_i that can be hold on a single physical machine (which means $v_i = \min_{i \leq I, r \leq R}\{\frac{c_i}{b_{i,r}}\}$).

As in equation 5.2.2, the number of VMs allocated on a physical machine is linearly related to each other. If non-integer VM numbers is allowed, the Pareto optimal VM capacity[3] would be the multi-dimensional plane formed by the linear combination of all the maximum number of VMs. Specifically, the Pareto optimal solution is, $\{\alpha_1 v_1, \alpha_2 v_2, ..., \alpha_I v_I\}$, where $0 \leq \alpha_i \leq 1, \sum_{i=1}^{I} \alpha_i \leq 1$. Hence, the Pareto optimal configuration for N physical machines would be $\{\alpha_1 v_1, \alpha_2 v_2, ..., \alpha_I v_I\}$, where $0 \leq \alpha_i \leq N, \sum_{i=1}^{I} \alpha_i \leq N$.

The goal of VM set selection for an application is to use a minimal number of physical machines to satisfy its resource demand. It is possible that not all the applications could be accommodated by the M physical machines when the workload exceeds the capacity of the cluster.

$$\min_{N \leq M} N \tag{5.2.3}$$

$$s.t. \sum_{i=1}^{I} b_{i,r} \times x_{i,k} \geq a_{k,r} \tag{5.2.4}$$

$$\sum_{k=1}^{K} x_{i,k} \leq \alpha_i v_i \tag{5.2.5}$$

$$0 \leq \alpha_i \leq N, \sum_{i=1}^{I} \alpha_i \leq N \tag{5.2.6}$$

5.2.3 Maximizing Overall Utility

The goal of the VM set scheduling is to maximize the overall utility of all the applications for the cloud provider, while still guaranteeing the resource demands and deadline requirements. We do not allow preemption among applications, hence we will know whether an application would be completed or not when it gets scheduled.

Let $x^*_{i,k,t}$ be the number of VM_i needed by application k at time t. $y_{m,i,k,t}$ is the number of VM_i of application k assigned to machine m at time t. We use D_k to denote the deadline of a deadline-sensitive application k. U_k is the utility function of application k. $\alpha_{i,m} v_i$ is the total number of VM_i accommodated on physical machine m. Then the VM scheduling problem can be modeled as follows:

[3]The Pareto optimal VM capacity means a VM combination in which it is impossible to increase the number of VMs for any type of without decreasing the number of VMs of another type

$$\max \sum_{k=1}^{K} U_k \tag{5.2.7}$$

$$s.t. \sum_{m \in M} y_{m,i,k,t} = x^*_{i,k,t} \quad \forall m \in M, t \in T \tag{5.2.8}$$

$$\sum_{k=1}^{K} y_{m,i,k,t} \leq \alpha_{i,m} v_i \quad \forall 0 \leq i \leq I, m \in M, t \in T \tag{5.2.9}$$

$$0 \leq \alpha_{i,m} \leq 1, \sum_{i=1}^{I} \alpha_{i,m} \leq 1 \tag{5.2.10}$$

$$\sum_{1<t<T-1} |y_{m,i,k,t} - y_{m,i,k,t-1}| \leq 2 \tag{5.2.11}$$

In this model, inequation (9) is the capacity constraint of physical machine i, inequation (11) is the nonpreemption requirement.

5.3 VM Set Selection and Scheduling

This section presents the greedy algorithm to select the VM set with minimum activated physical machines and a reservation-based VM scheduling algorithm.

5.3.1 VM Set Selection

We will first present the optimal VM set selection scheme and then prove its correctness. The optimal VM set selection algorithm is to greedily choose the VM set that will use the minimal number of physical machines for each application. We will prove in Theorem 1 and 2 that finding optimal VM set $\{x^*_{1,k}, x^*_{2,k}, .., x^*_{I,k}\}$ in a dynamic scenario can be transformed into the following linear programming problem.

$$x^*_{i,k} = \arg\min_{x_{i,k}} \sum_{i=1}^{I} \frac{x_{i,k}}{v_i} \tag{5.3.1}$$

$$s.t. \sum_{i=1}^{I} b_{i,r} \times x_{i,k} \geq a_{k,r} \tag{5.3.2}$$

$$x_{i,k} \in \mathbb{N} \tag{5.3.3}$$

Theorem 1 *$\{x^*_{1,k}, x^*_{2,k}, .., x^*_{I,k}\}$ is the VM set that can support the resource demand and use minimum number of physical machines for application k when only application k is considered in the system.*

Proof. The number of physical machines used by a VM set $\{\alpha_1 v_1, \alpha_2 v_2, ..., \alpha_I v_I\}$ would be $\lceil \sum_{i=1}^{I} \alpha_i \rceil$, which is the minimum integer no less than $\sum_{i=1}^{I} \alpha_i$. Since $\alpha_i = \frac{x_{i,k}}{v_i}$ and ceiling function is non decreasing, the optimal VM set for application k when only application k is considered in the system would be $\{x^*_{1,k}, x^*_{2,k}, .., x^*_{I,k}\}$. □

Theorem 2 *$\{x^*_{1,k}, x^*_{2,k}, .., x^*_{I,k}\}$ is the VM set that can support the resource demand and use minimum number of physical machines for application k when applications arrive as a stochastic process.*

Proof. If we can prove $\{x^*_{1,k}, x^*_{2,k}, .., x^*_{I,k}\}$ is the optimal VM set when there are two applications, then we can prove Theorem 2 without loss of generality. We denote the two applications as application 1 and application 2 and assume application 1 arrives earlier than application 2. In the naive case when application 2 arrives, application 1 already leaves the system then $x^*_{1,k}$ and $x^*_{2,k}$ would obviously be the VM set for the two applications with minimum activated physical machines. In a more complicated case, applications 1 and 2 coexist in the system for a while and we would consider this scenario from now on. In this case, the VM set with the minimal number of activated physical machines for each application would still be selected from the Pareto optimal VM sets (which can be easily proofed by contradiction). The optimal VM set for applications 1 and 2 would then be,

$$x^*_{i,1}, x^*_{i,2} = \arg\min_{x_{i,1}, x_{i,2}} \sum_{i=1}^{I} \frac{x_{i,1} + x_{i,2}}{v_i} \tag{5.3.4}$$

$$s.t. \sum_{i=1}^{I} b_{i,r} \times x_{i,1} \geq a_{1,r} \tag{5.3.5}$$

$$\sum_{i=1}^{I} b_{i,r} \times x_{i,2} \geq a_{2,r} \tag{5.3.6}$$

$$x_{i,a}, x_{i,b} \in \mathbb{N} \tag{5.3.7}$$

Since $\sum_{i=1}^{I} \frac{x_{i,1}+x_{i,2}}{v_i} = \sum_{i=1}^{I} \frac{x_{i,1}}{v_i} + \sum_{i=1}^{I} \frac{x_{i,2}}{v_i}$ and the constraints can separate, $\{x^*_{1,1}, x^*_{2,1}, .., x^*_{I,1}\}$ and $\{x^*_{1,2}, x^*_{2,2}, .., x^*_{I,2}\}$ are the optimal VM set for the application 1 and 2 respectively, which means the VM set with minimum activated physical machines can be selected separately.□

According to the above theorems, the VM set that satisfies the resource demand and uses minimum activated physical machines can be find by Algorithm 4. The complexity of VM set selection algorithm is $\Theta(1)$.

Algorithm 4 VM Set Selection

step 1: Define the Lagrange multiplier as
$L(x_{1,k}, x_{2,k}, ..., x_{I,k}, \lambda_1, \lambda_2, ..., \lambda_r) =$
$\sum_{i=1}^{I} \frac{x_{i,k}}{v_i} + \sum_{i=1}^{I} \lambda_i (b_{i,r} x_{i,k} - a_{k,r})$
step 2: Find the partial derivative $\nabla L = 0$
Which derives $\lambda_i^t = -\frac{1}{v_i b_{i,r}}, x_{i,k}^t = \frac{a_{k,r}}{b_{i,r}}$
step 3: The optimal VM set would be
$\left\{ \lceil \frac{a_{k,r}}{b_{1,r}} \rceil, \lceil \frac{a_{k,r}}{b_{2,r}} \rceil, ..., \lceil \frac{a_{k,r}}{b_{I,r}} \rceil \right\}$

5.3.2 VM Scheduling

After obtaining the optimal VM set of each application, we can use it as input and allocate the VMs to physical machines. On one side, the VM scheduling problem is NP-hard [83], where no polynomial time optimal solution is available by now. On the other side, the utility functions of applications decrease with time and scheduling time would be an overhead for the system. These concerns lead us to design a scheduling algorithm with fast scheduling process and yet provides acceptable performance.

Recall that the deadline-sensitive applications and best-effort applications have the 20/80 property and the deadline-sensitive applications have a more strict scheduling requirement. Hence we can use an accurate scheduling algorithm for the deadline-sensitive applications and a fast scheduling algorithm for the best-effort applications.

The scheme been adopted here is a reservation-based combined constrained programming and heuristic algorithm [29] which uses constrained programming scheduler for deadline-sensitive applications and a heuristic scheduler for best-effort applications. A dynamic portion of physical machines is reserved for the deadline-sensitive applications through time. We will describe the design specifications in this section.

5.3.2.1 CP Scheduler for Deadline-sensitive Applications

This chapter uses a constraint programming solver [50] for the deadline-sensitive applications, which is found to be 10+ times faster than the integer programming solvers for multi-resource scheduling problem [56, 78]. Since we do not allow preemption in our system, an application stays on the same machine during its execution. We can further speedup the scheduling process using this assumption: as describe in Algorithm 5 an application is excluded from the variable set once it has been allocated on a physical machine.

5.3.2.2 Heuristic Scheduler for Best-effort Applications

This chapter uses maximum residue as the heuristic scheduler for best-effort applications. Let's define the maximum residue scheduler as selecting the physical machine m that satisfies the following expression,

$$\arg\max_{m} \sum_{r=1}^{R} [l_{m,r}(b_{i,r} \sum_{i}^{I} x_{i,k}^{l})] \tag{5.3.8}$$

where $l_{m,r}$ is the remaining number of type r resource (CPU, memory, storage, etc.) that can be hold on physical machine m. $x_{i,k}^{l}$ is the remaining number of type i VM that haven't been allocated of application k. The heuristic uses dot product to capture the similarity between the remaining VM capacity vector of the physical machine and the remaining VM demand vector of the application. The VMs of an application is assigned to the physical machine with a residual capacity that is closest to the application's remaining VM demand.

The combined constraint programming and the heuristic scheduling algorithms are presented in Algorithm 5 and Algorithm 6. To schedule pending jobs, we adopt a basic deadline-based approach as in Algorithm 7.

Algorithm 5 Combined CP and Heuristic Algorithm

When {application k arrives}
if $k\_is\_deadline\_sensitive$ **then**
    Heuristic_scheduler(k); assign k to an unreserved machine
else
    CP_scheduler(); assign a reserved machine to application k
    if CP failed to assign a reserved machine to k **then**
        $Heuristic\_scheduler(k)$; assign k to an unreserved machine
    end if
end if
When {an application k running on machine m finishes}
Release machine m resources for k
$Schedule\_a\_queued\_job(m)$

5.3.2.3 Cluster Reservation for Deadline-sensitive Applications

To better guarantee the deadline requirements for deadline-sensitive applications, this work reserves and dedicates part of the physical machines to accommodate them. When a

Algorithm 6 Heuristic_scheduler(k)

Symbols: k – job to be scheduled
for Machine $m \in$ Unreserved_cluster **do**
    Find min $\left\{ \frac{m_r}{m_k} + \frac{c_r}{c_k} \right\}$ $m_r \geq m_k, c_r \geq c_k$
end for
Assign application k to machine m
if *No_unreserved_machine_available* **then**
    Insert application k to the deadline-ordered queue Q
end if

Algorithm 7 Schedule_a_queued_job(m)

Symbols: m – available machine
for Application $k \in$ job queue Q **do**
    if $Capacity(m) \geq Demand(k)$ **then**
        Assign application k to machine m **return**
    end if
end for

deadline-sensitive application arrives, it is scheduled (using CP) to one of the reserved machines if there are any left. If no reserved machines is available, it will be scheduled to an unreserved machine using the heuristic algorithm. We use Algorithm 8 to dynamically determine the reservation threshold. The more physical machines are reserved, the higher probability that a best-effort application will get rejected. On the other side, more physical machines are provided for general use, more deadline-sensitive applications would be scheduled with the less accurate heuristic scheduler. An appropriate reservation threshold is important to maintain a high utilization rate for the cluster.

5.4 Evaluation

This work evaluates the VM selection and scheduling algorithm in this section using trace-driven simulations. Google Cluster Trace [49, 59] is a public cloud trace from a heterogenous workload cloud with around 12,000 machines for a month long period. This work uses Google Cluster Trace since it contains both resource demand trace from each application and resource usage trace from the physical machines. Resource demand and usage are normalized to the largest capacity of the resource on any machine in the trace (which is 1.0). The resource demand trace contains 500 files of application resource demands (in terms of CPU, memory, storage), each with around 5000 applications. Note that an application usually contains multiple tasks and each task has a resource demand. However, in this work

Algorithm 8 Reservation_Threshold(k)

Symbols: θ – current reservation threshold, ε – adjustment threshold, θ_0 – initial threshold get by tuning, η – adjustment factor get by tuning, q_d, q_b – accumulated weighted resource demand for deadline-sensitive and best-effort applications respectively, q_k – weighted resource demand of application k
When {application k arrives}
if $k\_is\_deadline\_sensitive$ **then**
    $q_d \leftarrow q_d + q_k$
else
    $q_b \leftarrow q_b + q_k$
end if
if $\frac{q_d}{q_b} < \theta_0 - \varepsilon$ **then**
    $\theta \leftarrow \theta - \frac{\eta q_k}{q_b}$
else if $\frac{q_d}{q_b} > \theta_0 + \varepsilon$ **then**
    $\theta \leftarrow \theta + \frac{\eta q_k}{q_b}$
end if

we do not dive into the granularity of tasks and will sum up the resource demand of the tasks into the resource demand of an application. Each application has a field indicating its priority in terms of high, middle and low. We define the application with high priority as deadline-sensitive applications and the rest as best-effort applications. Since the trace doesn't have a deadline field for the applications, we add a deadline for the deadline-sensitive applications by ourselves. We first add the deadline as 10% of an application's duration, and will vary the deadlines and see their effects in later evaluations. We would also vary the utility functions for best-effort application to see their effects in the following part. We assume 6 primary VM instances listed in Table 5.2. The configurations of the VMs is proportional to the VM instances provided by [45]. We will use part of these VM settings in our later simulation. The constraint [57] of the Gecode scheduler been used is the "cumulatives" constraint provided by the multi-resource scheduling package [50]. The simulation is done on a workstation with quad-core 2.0GHz processor and 16GB RAM.

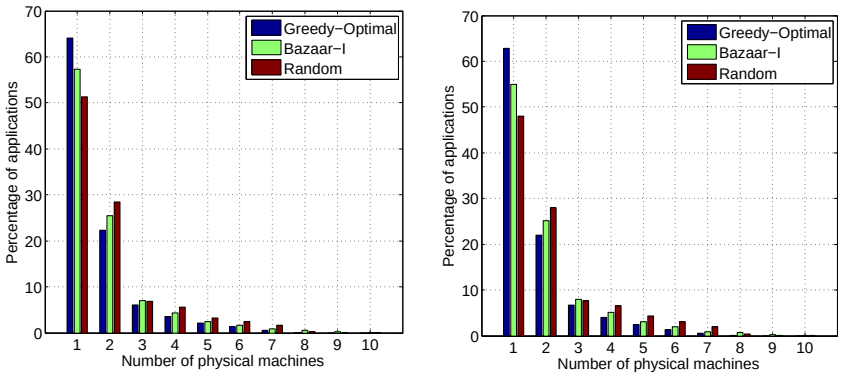
5.4.1 VM Set Selection

We compare our VM set selection scheme with the resource selection specified in [71] and the random selection approach (from the Pareto optimal VM sets). The number of physical machines used by the three approaches is shown in Fig. 5.1.

As can be seen from the figures that the greedy-optimal VM set selection algorithm will

Table 5.2: VM configurations (normalized to).

Category	CPU	memory	storage	price
m3.medium	0.0625	0.0625	0.0125	0.067
m3.xlarge	0.25	0.25	0.25	0.266
c3.large	0.125	0.0625	0.1	0.105
c3.4xlarge	1	0.5	1	0.84
r3.large	0.125	0.25	0.1	0.175
r3.4xlarge	1	1	0.5	0.7



(a) With VM 1-6.

(b) With VM 1,2,4,6.

Figure 5.1: Number of physical machines needed using each VM set selection scheme. Greedy-optimal is our VM selection in Section III.

use less physical machines than random VM selection within feasible VM set or the proportional resource selection in [71] in both two VM configurations. From now on, we will use all VM configurations in Table 5.2.

Note that the random selection approach would need to compute the Pareto optimal VM sets. The time complexity is $\Theta(\prod_{i=1}^{I} g_{k,i})$ where $g_{k,i}$ is the number of using VM_i alone to support application k. Hence, the random selection would be very time costly when the resource demand grows large.

5.4.2 VM Scheduling

We compare our combined VM scheduling algorithm with several scheduling approaches, namely first-fit, best-fit and Bazaar-I with various forms of utility functions for the best-

effort applications. For best-fit and first-fit scheduling, we will use both greedy-optimal and random scheme as the VM selection algorithm.

5.4.2.1 Utility Functions

We evaluate the VM scheduling algorithms using two utility functions, namely, the inverse proportional function and the linear drop function as shown in Fig. 5.2.

The inverse proportional function is defined as the profit of the application when the time after its arrival is within its duration and then will drop by an inverse proportional function. Specifically, the utility function for the best-effort applications is $\frac{p_k}{1+\min\{E_k-D_k,0\}}$, where p_k is the profit of application k, D_k and E_k are the duration and existing time for application k respectively.

The linear drop function is defined as the profit of the application when the time after its arrival is within its duration and then will drop by a linear function. Specifically, the utility function for the best-effort applications is $\max\{p_k - \min\{E_k - D_k, 0\}, 0\}$.

5.4.2.2 Completion Rate

Completion rate is an important factor to indicate the fulfilment of user SLAs. Fig. 5.3 and Fig. 5.4 shows that our combined algorithm can achieve higher completion rate of both deadline-sensitive applications and best-effort applications than Bazaar-I, best-fit and first-fit policies using both random and greedy-optimal as the VM set selection algorithm with 2048 physical machines under the inverse proportional utility function.

As can be seen from both figures, the proposed combined algorithm can improve the overall completion rate, especially for deadline-sensitive applications. This result reflects the fact that deadline-sensitive applications can use both the reserved resources and the unreserved resources.

Fig. 5.5 and Fig. 5.6 shows the comparison of the four approaches under the linear drop utility function. The completion rate of the best-effort applications is a little lower w.r.t. the inverse proportional case and the completion rate for the deadline-sensitive applications increases. We can see from the figures that the inverse proportional utility function pays more attention to the deadline-sensitive applications than linear drop utility function.

Fig. 5.7 shows the completion rate of the deadline-sensitive and best-effort applications w.r.t. the cluster scale using our VM selection and scheduling algorithm. The original Google Cluster Trace contains around 12000 physical machines to record the usage data.

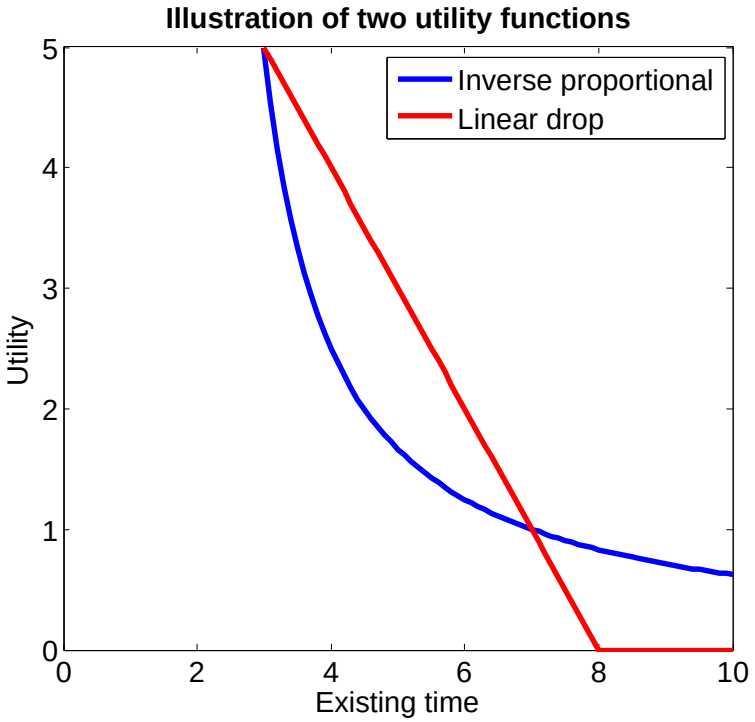
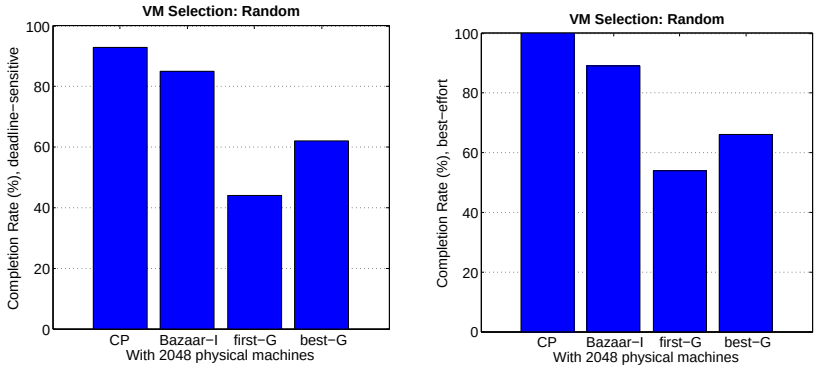


Figure 5.2: Two representative utility functions for best-effort applications.



(a) Deadline-sensitive Applications. (b) Best-effort Applications.

Figure 5.3: Completion rate for inverse proportional function with 2048 physical machines and random VM selection for best-fit and first-fit.

Table 5.3: Utilization with different cluster scales for the two utility functions.

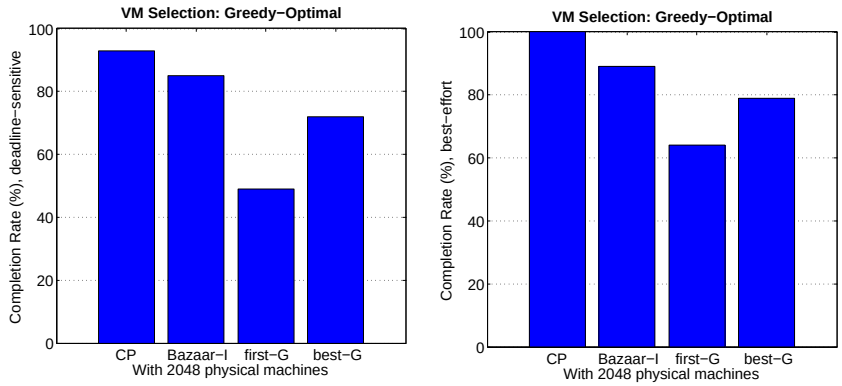
Nr. of PMs	128	256	512	1024	2048
Inverse Proportional	93.0%	92.9%	92.6%	92.8%	92.6%
Linear Drop	93.0%	92.7%	92.5%	92.6%	92.4%

However, we can see that 4000 machines would almost be suffice to hold all applications using our scheduling scheme.

5.4.2.3 Resource Usage

Fig. 5.8 shows the averaged CPU utilization ratio of the activated (have a non-zero workload) physical machines for the four schemes in a cluster of 2048 physical machines under the inverse proportional utility function. As can be seen from the figure, the proposed solution will improve the utilization ratio of the activated machines.

The averaged resource usage of activated physical machines of our VM selection and scheduling schemes with scales is shown in Table .5.3. As can be seen from the table, our VM selection and scheduling algorithm will keep a high utilization ratio for various cluster scales. We also find that using inverse proportional as the utility function will provide a little higher resource utilization ratio since it will still try to schedule the best-effort application with long waiting time when there are available physical machines.



(a) Deadline-sensitive Applications. (b) Best-effort Applications.

Figure 5.4: Completion rate for inverse proportional function with 2048 physical machines and greedy-optimal VM selection for best-fit and first-fit.

Table 5.4: Comparison of overall utility with 2048 physical machines and inverse proportional utility functions.

Category	Inverse Proportional	Linear Drop
CP	1	0.9872
Bazaar-I	0.9144	0.9089
first-G	0.5382	0.5426
best-G	0.7809	0.7801

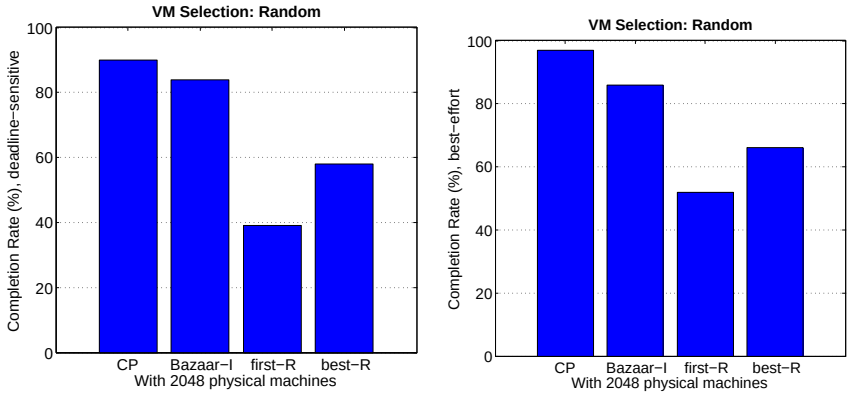
5.4.2.4 Overall Utility

The overall utility shows a similar trend with the completion rate. We show our result in Table. 5.4. Note that the overall utility is normalized to our greedy-optimal and combined constrained programming solution under inverse proportional utility function for comparison purposes. For all four schemes, the overall utility by using inverse proportional and linear drop are almost the same.

5.4.2.5 Scheduling Speed

Table. 5.5 compares the VM scheduling speed and its accuracy of our combined algorithm with pure constraint programming on one of the 500 traces.

As can be seen from the table, although CP can achieve higher accuracy, the time com-



(a) Deadline-sensitive Applications. (b) Best-effort Applications.

Figure 5.5: Completion rate for linear drop function with 2048 physical machines and random VM selection for best-fit and first-fit.

Table 5.5: Our combined algorithm vs. pure CP.

Category	Constraint Programming	Combined Algorithm
Computation time for scheduling	1775min	12min
Overall utility	1	0.76

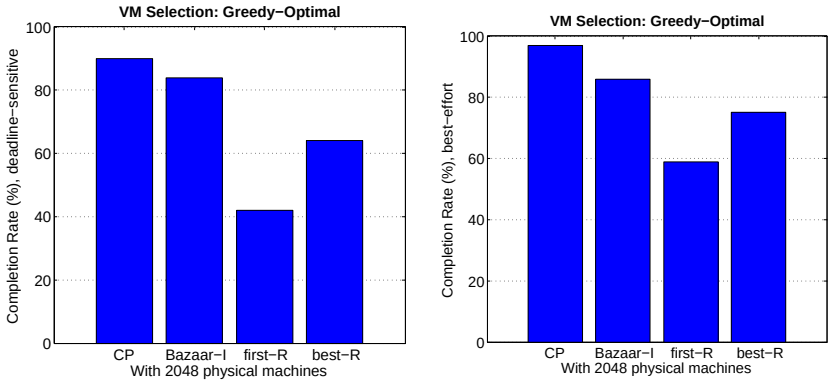
plexity makes it impractical for online use. Our combined algorithm on the other hand, can provide reasonably good results within a short amount of time for scheduling.

5.4.2.6 Extending Deadlines

We extend the deadlines for the deadline-sensitive applications to see their effects on completion rate. We tested for a cluster with 2048 physical machines and the result is shown in Fig. 5.9.

5.4.3 Reservation Threshold

The initial reservation threshold and the adjustment factor are learned from the training process using $\frac{1}{5}$ of the cluster data. We use the first 100 files of the resource demand trace to determine the initial reservation threshold and the second 100 files to determine the adjustment factor.



(a) Deadline-sensitive Applications. (b) Best-effort Applications.

Figure 5.6: Completion rate for linear drop function with 2048 physical machines and greedy-optimal VM selection for best-fit and first-fit.

Table 5.6: Reservation Threshold with 2048 physical machines.

Nr. of Traces	100	200	300	400	500
Estimated Threshold	-	73.6%	74.2%	74.6%	74.7%
Calculated Threshold	73.4%	74.2%	74.8%	74.5%	75.1%

The initial reservation threshold is computed by changing the reservation threshold from 0 percent to 100 percent, calculate and record the overall utility for the two utility functions and then find the ratio that will render the highest overall utility. The process to determine the initial threshold for the two utility functions is shown in Fig. 5.10. The overall utility is also normalized to its maximum value for simple comparison. In both cases, the overall utility increases with the reservation threshold at first and drops afterwards. This is consistent with our explanation of reservation threshold in Section III.

To get the adjustment factor, we first calculate the best reservation threshold for the first 120, 140, 160, 180 and 200 trace files, and denote it as $\theta_1, \theta_2, \theta_3, \theta_4$ and θ_5. Then the adjustment factor η is calculate as $\eta \leftarrow \frac{1}{5}\sum_{i=1}^{5} |\theta_i - \theta_{i-1}| \frac{q_{bi}}{q_i}$, where q_{bi} and q_i is the accumulated weighted resource demand for deadline-sensitive applications and application i respectively. We evaluate the threshold estimation a comparison of the estimated threshold and the calculated best reservation threshold in Table 5.6 and Table 5.7.

As can be seen from Table 5.6 and Table 5.7 the estimated threshold is very close to our calculated threshold with various cluster scales and reservation threshold. Hence with the initial reservation threshold and adjustment factor we can provide an accurate estimation of the optimal reservation threshold.

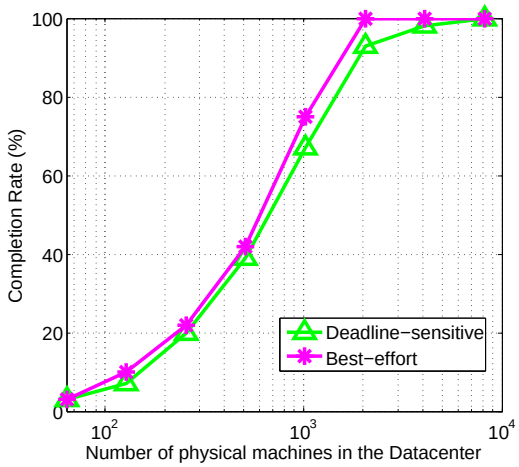


Figure 5.7: Completion rate w.r.t. different cluster scales under inverse proportional utility functions.

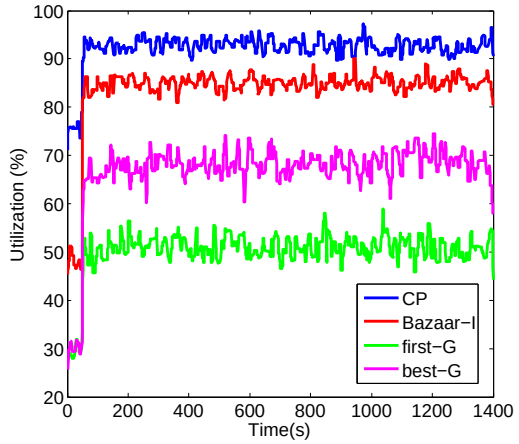


Figure 5.8: CPU utilization ratio with 2048 under inverse proportional utility function.

Table 5.7: Reservation Threshold with 1024 physical machines.

Nr. of Traces	100	200	300	400	500
Estimated Threshold	-	79.8%	79.5%	80.9%	81.0%
Calculated Threshold	80.2%	79.3%	79.8%	81.5%	81.1%

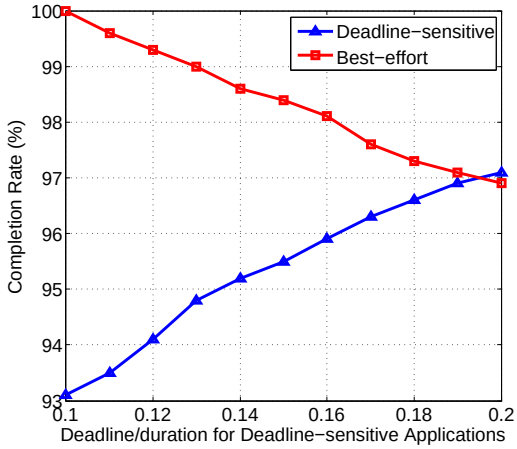


Figure 5.9: Completion rate w.r.t. different deadlines under inverse proportional utility function in a 2048 physical machine cluster.

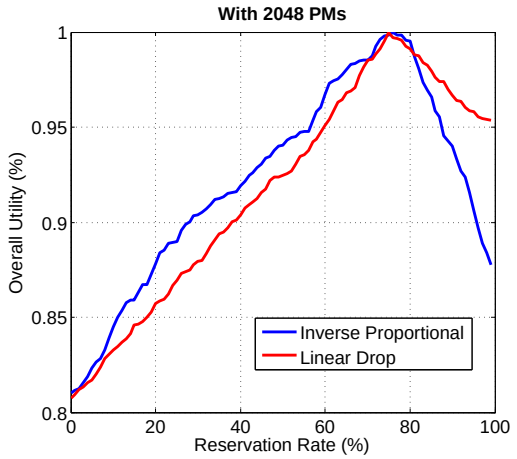


Figure 5.10: Initial reservation threshold.

The adjustment threshold ε will affect the stability of the system in terms of adjustment frequency. If ε is too small there will be frequent changes between consecutive scheduling decisions; on the other hand, if ε is too large, the estimated threshold will depart from its true value. In this paper, we define the adjustment threshold to be 0.01.

A number of existing works study the VM scheduling problem in cloud environment. The basic VM scheduling problem is related to the multi-dimensional bin packing problems. The primary goal is to use a minimal number of bins to hold all the incoming items with jobs arrive and leave in a stochastic manner. The VM scheduling with random arrive and departure time is NP-hard that has no polynomial time optimal solution by now [86]. A summary of heuristics of the multi-dimensional stochastic bin packing problems can be find in [75]. [72] addresses a joint routing and VM placement problem which extends the bin packing constraints to account for the communication between VMs. [55] gives a good discussion of recent multi-dimensional bin packing problem for theoretical references. However, the stochastic bin packing problem only considers the scheduling from VM to physical machines.

There are also literature works address the dynamic VM scheduling problem in a practical way. Yanagisawa et al. [94] proposed an allocation scheme that assigns VMs to physical machines based on the fluctuated CPU demand using mixed integer programming. They partitioned the time-horizon into intervals and allocate the resource periodically to reduce the scheduling time. [29] proposed the combined constraint programming approach for VM scheduling with fixed deadlines for each applications. [73] considers maximizing the throughput of a queueing system with a finite number of servers in asymptotic way. The problem with these three approaches is they only consider the VM scheduling problem however there exists multiple feasible VM sets. Jalaparti et al. [71] proposed the idea of user-oriented resource provisioning. They designed an online scheduler that allows the cloud provider to choose among feasible VM sets to improve the utilization for MapReduce jobs. Wieder et al. [85] also aims to select the appropriate VM set for MapReduce jobs, so that users can specify their goals, such as minimizing monetary cost or completion time. The problem with these two approaches is that they use a heuristic VM set selection algorithms, hence would use more physical machines compared to the optimal solution. When the system is overloaded, this would cause low completion rate of the applications. [70] proposed a deadline-aware VM allocation scheme modeling job utilities as a function of completion time. The scheme aims to maximize the minimum utility across all jobs while the utilization and profit of the cloud provider is not considered.

In contrast to these works, our work not only allows users to specify their SLA requirement but also allow various SLAs according to the applications' characteristics. To the best of our knowledge, we are also the first to propose the optimal VM set selection algorithm which minimizes the number of physical machines for each application.

5.5 Chapter Summary

This chapter demonstrates that multiple VM sets can satisfy the application's resource demand and the cloud provider could leverage this property to improve its resource utilization rate and without violating the applications' SLAs. We propose a VM set selection and scheduling approach using combined constraint programming and reservation-based scheme. We dynamically determine the reservation threshold for deadline-sensitive applications.

Future work includes refining our model to account for the energy consumption for providing datacenter services. The energy consumption for the commodity physical machines is not proportional to its workload, and the higher the utilization the more energy efficient it would be. Hence, our goal to improve the utilization is consistent with saving energy. Another issue is to incorporate the networking cost between VMs of the same application to the VM scheduling problem. The problem could also be modeled into the mixed integer programming form. The difficulty lies in transforming the new model into the constraints provided by the CP solver.

Chapter 6

Mobile Cloud Resource Scheduling

Leveraging remote computing resources to execute part of the computation workload from mobile device could improve its performance (in terms of time and scope of applications) and battery life. The underlying rationale is that the speedup by using more powerful remote computation resources could potentially overcome the communication delay and scheduling time overhead. The goal of the scheduler is to improve the performance and battery life of mobile devices while at the same time maximizing the cloud provider's profit. There are two aspects of mobile cloud resource scheduling problem.

1. **Online code partition and offloading**. Not all parts of the mobile application should run on the cloud side. Some parts of the application are unremoteable and could only run on the mobile side (such as user interactive components, sensor related components, etc.). The partition decision is also relevant to the current networking status. The scheduler needs to make the partition decision at runtime w.r.t the latency and bandwidth status.

2. **Resource scheduling on cloud side**. When there are multiple mobile users requesting the cloud resource at the same time, the cloud resources are shared by the offloaded mobile application requests. The cloud scheduler also needs to make the resource allocation decisions among the submitted mobile requests. The cloud side scheduling also needs to maintain the mobile side SLA and to improve the cloud side utilization rate.

This chapter presents an efficient code partition algorithm with high partition accuracy and a cloud resource scheduler that can improve the resource utilization. Our design is motivated by the clustered offloading property of the call graph, which is based on the observation that when a method is offloaded, the subsequent invocations will be offloaded with a high chance. Hence, instead of making an offloading decision for each method as in the 0-1 ILP approach, this work proposes a solution that determines the best offloading

and integrating points, which are several starting and ending methods pairs that would have a minimal cost if all the execution sequence in between are offloaded. This work proposes a time complexity code partition algorithm using depth-first search (DFS) and a linear time searching scheme, where is the number of remoteable methods and is the number invoking relations in the application. This work evaluates the proposed scheme using two mobile applications, namely face recognition and natural language processing. Experiments results show that the partition algorithm is 2 orders of magnitude speed up for the face recognition application and the natural language processing application than the 0-1 ILP approach with more than 90% accuracy.

This chapter is based on the work in [89].

6.1 Background and Motivation

A new class of cognition augmenting applications for smartphones such as face or object recognition, image or video editing, natural language processing and augmented reality is emerging and attracts increasing attention [63, 64, 69]. This kind of applications require a large amount of computation, storage and energy which cannot be satisfied by off-the-shelf mobile devices [87]. While more sophisticated hardware can be adopted to carry out these application tasks, its usage will be limited by its size, weight and monetary cost. An alternative way of facilitating augmented cognition applications on mobile devices is to leverage cloud service or nearby infrastructure to support part of the execution. When the connection quality to the server is good, offloading some execution to the cloud could be beneficial. Given the ubiquitous deployment of WiFi/3G networks [76, 77], the ongoing development of 4G systems [77], and the availability of commercial cloud infrastructure, the future of mobile cloud computing is promising.

More and more mobile computing devices are enjoying the advanced services and applications offered by servers and more recently the cloud infrastructure, ranging from text and image editing, games, to virus detection, natural language processing, or scientific computing. Each of these applications or jobs require different resource demands, and has different service level agreement (SLA) requirements in terms of response time or completion time. For example, interactive applications like image editing or voice detection require a tight response time, typically a few seconds. Some production jobs, like virus detection may have a more relax SLA requirements, where users may only require an overall completion time delay. However, in some cases, no specified completion time delay is provided; instead, the main concern is to get the job done within a overall budget. In fact, various literatures have shown that the workloads in public clouds are quite diverse in terms of priority, SLA requirement, resource requirement, and job duration [53, 74].

To execute the code on mobile device in mobile cloud computing paradigm, the mobile device would offload part of the code from remote infrastructure. A key issue here is to determine what's the most energy- and/or time-efficient way of allocating the components of the target application given the current network condition and cloud resource usage status, which is also known as the code partition problem [10], [23], [9], [22]. While cloud infrastructure provides high computation and service capacity, transferring the input and return state is time and energy consuming.

In extreme cases, when the wireless connection is too weak or the amount of state needed to be transferred is too big, executing code on cloud can impair the performance and waste energy. Hence, before actually running its code on the cloud environment, an offloading decision should be made for the components of the target application with regard to the current network condition and device capacities. We follow the mobile cloud system archi-

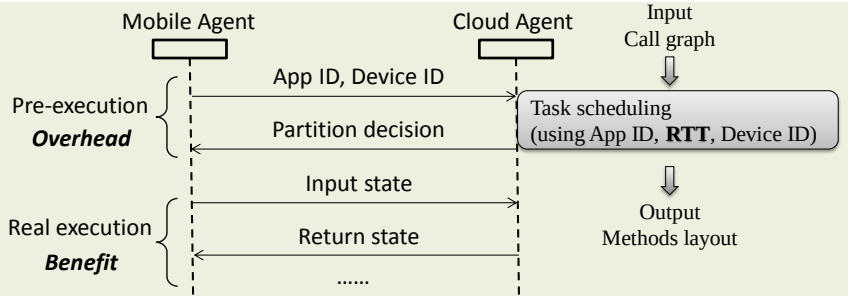


Figure 6.1: Process of mobile cloud computing and the role of code partition.

tecture [10] shown in Fig. 6.1. As can be seen from the figure, the mobile cloud computing system usually contains two agents, one on the smartphone and one on the cloud. Code partition process is conducted before the real execution on the server based on the application identifier, device identifier and the RTT between the smartphone to the cloud. This kind of information provides the server with the knowledge of program complexity, hardware capacity and network conditions. After getting the partition layout, the smartphone would transfer the necessary state to enable remote execution. Even though a good partition layout can improve the performance, the decision making process itself is an overhead of the real computing 6.2. Thus, the code partition algorithm should be both accurate and fast. Furthermore, a good code partition algorithm should also have the following characteristics:

1. **Real time adaptability**. The partition algorithm should be adaptive to network and device changes. For example, an optimal partition for a high bandwidth low latency network and low capacity client might not be a good partition for a high capacity client with bad network connection. Since the network condition is only measurable at run time, the code partition algorithm should be a real time online process.

2. **Partition efficiency**. At the early stage of mobile cloud computing, researchers used to adopt the full Virtual Machine (VM) migration approach, in which the mobile device serves as a dumb client that receives and renders data [35, 36]. This approach is gradually discarded by current designs [10, 23], and most recently approach [82], which generally take a more fine-grained perspective (typically at a method-level granularity) and only migrate computation intensive parts of the application. For simple applications (e.g., an alarm clock), making code partition decisions for these methods at real time is not difficult. However, some popular applications (e.g., speech/face recognition) are quite complex and contain many methods. Therefore, a highly efficient algorithm that can perform real-time code partition for applications with a large number of methods is demanded.

On the other hand, since multiple mobile users may request service at the same time, it

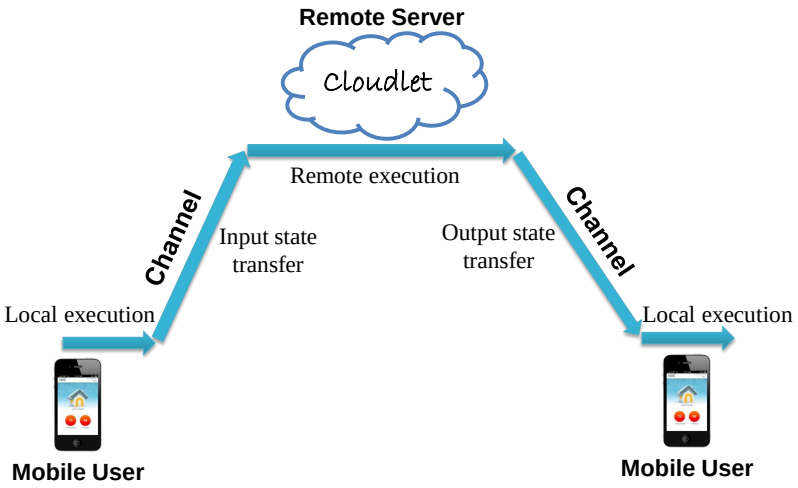


Figure 6.2: Overall performance in terms of end-to-end delay.

would be possible that the cloud infrastructure would exceed its resource capacity. Hence, in addition to carefully schedule the job requirements in the cloud computing infrastructure, a code partition decision should be made for the components of the target application w.r.t the current network condition and resource usage status.

Despite the need of strict response time by mobile cloud services, the current VM provisioning scheme in public cloud is not designed for the mobile cloud offloading scenario. The average boosting time is about 27 seconds in Amazon [82], and minimum unit of purchasing an VM/instance is an hour. However, the mobile applications need immediate response and usually takes several minutes. Hence, to use the cloud resources for mobile cloud applications, new resource multiplexing scheme should be proposed at the cloud side.

6.2 Call Graph Model

The interaction relationship between the mobile device and the cloud infrastructure can be represented by a call graph model. A call graph is a directed weighted graph. The vertices are methods of the application, while the edges represent the calling relationship from caller to callee [89]. As describe in [89], each vertex and each edge has two weights, where the weights of a vertex are the cost of executing the method on the infrastructure and on the mobile device, respectively, and the weights of an edge are the cost of transferring the input and return states if the caller and callee methods are executed on different places. In

order to correctly execute an application on two separate machines, the system needs to transfer all potential states that could be accessed by the callee or caller. Specifically, the input state is the input parameter plus the public state which includes the current object's member variables (including nested member variables), the static classes, and the public static member variables. The return state is the return value plus this public state.

For a given application, the execution cost is affected by input parameters and device characteristics, while the transfer cost is related to the input and network condition. The metric of cost can either be time or energy consumption. Our partition algorithm is applicable to both metrics. However, without loss of generality, we adopt time as the cost metric hereafter because the profiling process does not need any extra device (e.g., a power meter). Given that the computation capacity of the cloud computing infrastructure is stronger than that of the mobile device, we assume that the execution time on the infrastructure is smaller than that on the mobile device. This assumption still holds when using energy consumption as the cost metric, where only the cost on the mobile device is taken into account.

Not all methods can be executed remotely [9, 10]. For example, methods that interact with the user interface or access I/O devices and local hardware should not be offloaded (e.g., GPS, accelerator or other sensors). In addition, remote execution might raise security threats. In such cases, offloading should be avoided. Automatic classification of suitability for remote execution is possible using static analysis [9].

6.3 Joint Partition and Scheduling Model

The joint code partition and scheduling model is provided in this section. It is worth to mention that the mobile cloud computing model differs from our previous work [88] in that *it satisfies both the cloud computing provider and the mobile user*. The goal of the cloud provider is to maximize its income with fixed number of machines. For the mobile user side, the goal is to meet his or her SLA requirement. To achieve these goals, a new model is needed to describe the mobile cloud computing system. Similar to public cloud service, to achieve a fair, profitable, and efficient utilization of the available resources at the cloud computing infrastructure, we consider the multi-resource pricing policy for mobile cloud computing. Since the pricing policy is orthogonal to our resource scheduling problem, we consider two widely-adopted pricing policies in literature [26, 27, 29, 84]: *Market Pricing* and *Linear Pricing*.

Market pricing adopts current cloud operators's policies [48]. The Google Compute Engine provides three VM types, namely standard, high memory and high CPU. The configurations that are not covered by the policy are calculated by interpolation. In this pricing

policy, the price grows nonlinearly with memory usage.

Contrast from market pricing, linear pricing is often adopted by researchers to evaluate their for simplicity [84]. The baseline we use is the standard VM type in Google Compute Engine [48] where the price will increase by 0.104 if the number of virtual cores increases by 1.

In this paper, we allow the flexibility that the cloud computing infrastructure provider can adopt either the market pricing policy or linear pricing policy. Then the joint code partition and resource scheduling problem in cloud computing can be modeled as a mixed integer programming problem with an objective to maximize the overall revenue of the infrastructure provider:

$$\max \sum_{j \in J_1 \cup J_2} P_j \tag{6.3.1}$$

$$s.t. \sum_{j \in J_1 \cup J_2} \mathbf{R}_{j,t} \times x_{j,m,t} \leq \mathbf{C}_m \quad \forall m \in M, t \in T \tag{6.3.2}$$

$$T_{v,r} = h\left(\mathbf{R}_{v,r}\right), \forall v \in V_i, j \in J_1 \cup J_2 \tag{6.3.3}$$

$$P_j = \sum_{v \in V_j} I_v \times T_{v,r} \times g\left(\mathbf{R}_{v,r}\right), j \in J_1 \cup J_2 \tag{6.3.4}$$

$$\sum_{v \in V_j} [I_v \times T_{v,r} + (1 - I_v) \times T_{v,l}] + \sum_{(u,v) \in E_j} \mid I_u - I_v \mid \times T_{u,v} \leq D_j, \forall j \in J_1 \tag{6.3.5}$$

$$P_j \leq B_j \forall j \in J_2 \tag{6.3.6}$$

$$x_{j,m,t} \in \{0,1\} \tag{6.3.7}$$

$$\sum_{m \in M} x_{m,j,t} \in \{0,1\} \quad \forall j \in J, t \in T \tag{6.3.8}$$

where J_1 is the set of jobs without budget limit, J_2 denotes the set of budget-constrained jobs. P_j is the income of having job j running on the cloud data center. The total time horizon is T. For a time slot $t \in T$, $\mathbf{R}_{j,t}$ denotes the resource demand of job j at that time slot. We consider a multi-resource scenario here, thus $\mathbf{R}_{j,t}$ is a vector of multiple resource types. $\mathbf{C}_m$ is the resource capacity of machine m, which is also a vector. Notice that we do not impose the homogeneity property on the resource capacity of machines. That is, different machines can have heterogenous resource capacities. For the code partition of job j, $v \in V_j$ is a flexible method that can be downloaded from the cloud data center. I_v is a binary variable indicating that the corresponding method is running remotely (i.e., $I_v = 1$) or locally (i.e., $I_v = 0$). $\mathbf{R}_{v,r}$ is the allocated resource for method v on the remote side, $T_{v,r} = h(\mathbf{R}_{v,r})$ is the execution time of running method v on the cloud data center using resource $\mathbf{R}_{v,r}$. $T_{v,l}$ is the execution time of running method v on the mobile user locally.

$T_{u,v}$ is the time overhead for invoking parameter exchange if two interactive methods u and v are not running on the same location. $g(\mathbf{R}_{v,r})$ is the time-unit pricing function to charge the method v for utilizing the resource $\mathbf{R}_{v,r}$ from the cloud data center. $x_{m,j,t}$ is a binary scheduling variable, where 1 indicates job j is assigned to machine m at time t and 0 otherwise. For jobs with completion time requirements (i.e., D_j), the overall execution and data transmission time should not exceed its completion time. For jobs with budget limit (i.e., B_j), its payment should not exceed its budget. Similar to [88], for any physical machine, the resource usage should not exceed its capacities at anytime.

Note that if the workload exceeds the infrastructure's capacity, not all jobs would get allocated in the infrastructure. It is possible that the jobs that do not have a specified completion time would be postponed. Furthermore, jobs might fail to finish within their specified completion times and violate their SLAs. In this case, they should not contribute to the overall revenue of the service provider. We assume, jobs can be split such that no job requires more than 1 physical machine to execute.

As can be seen, this model differs from our [88] work in that it considers the influence of code partition towards resource scheduling which is equation 3 to 6.

6.4 Partition and Allocation Algorithm

When all methods are executed on the cloud data center, the joint code partition and resource scheduling problem will reduce to the SLA-aware multi-resource scheduling problem, which is NP-hard [33]. Hence, the original code partition and resource scheduling problem is also NP-hard. In mobile cloud computing, no user specified resource demand is available, hence existing resource scheduling algorithms do not apply here. In the meanwhile, previous code partition [9, 10] algorithms, including our own approach in [89] aim to minimize the overall time or energy consumption rather than get the specified resource demand based on a user provided SLA code partition algorithm needs to be developed.

In this section, we first conservatively estimate the resource demand by leveraging a polynomial time code partition algorithm in our previous work [89]. After obtaining the resource demand, the jobs are allocated to physical machines using a reservation-based scheduling scheme as describe in our previous work in [88], where the resource demand is used as the input for resource scheduling.

6.4.1 Code Partition

In mobile cloud computing, the mobile users do not explicitly specify their resource demand as in normal resource scheduling problems. Instead, an SLA requirement is specified for each job. To get the user requests allocated, the first thing is to convert the SLA requirement into specified resource demand for the completion time-aware jobs, we then illustrate how to deduce the resource estimation algorithm for completion time-aware jobs to budget limit jobs.

In the following part, we illustrate the resource estimation scheme for completion time-aware jobs leveraging an optimal polynomial time code partition algorithm. This resource estimation algorithm differs from our previous work [89] in two ways:

1. the code partition algorithm here is to get a feasible partition and its corresponding resource demand, while in [89], the goal is to get a partition that would minimize the overall time.

2. the code partition algorithm here works on a general call graph while in our previous work [89], we need to convert the call graph into a call link.

3. the code partition algorithm provided here is an optimal polynomial time algorithm while the algorithm in [89] is a heuristic one.

In the following part, we first proposed a general property on a call graph, so called the minimal download property. Using that property we can obtain a polynomial time code partition algorithm with the goal to minimize the overall time. To estimate the resource demand, we use a binary search approach to conservatively find the resource demand.

6.4.1.1 Minimal Offload Property

We will illustrate the minimal offload property in the following part. Note that in [89], we have proposed a similar one time offload property. The difference resides in two folds:

1. the minimal offload property works for a general call graph rather than a coverted call link. Hence, not only the first part of the minimal offload property works on the different context, the second part of the minimal download property never exists in our previous work.

2. the code partition algorithm provided here is an optimal polynomial time algorithm while the algorithm in [89] paper is a heuristic one.

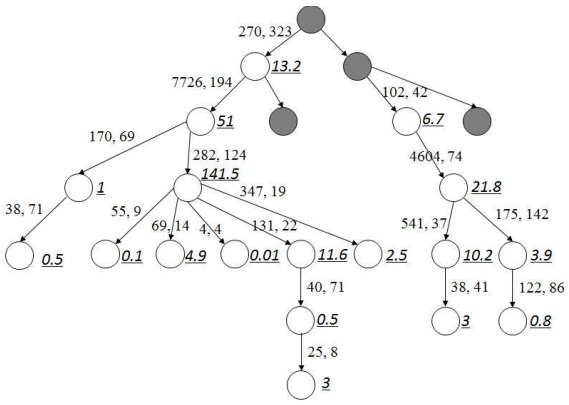


Figure 6.3: Call Graph for a face recognition application. A 42kB jpeg is recognised among a set of 32 images of same size. White vertices (in contrast to grey ones) indicate remotely executable methods; The number on the right hand side of a method with underline and italics format is its execution time on server side with the minimal amount of resource. Vertices are labelled with million CPU cycles of methods; Edges are labelled with sizes of input ant return states.

As illustrated in [89], Fig. 6.3. shows the code partition result of the face recognition App to achieve the minimal overall time using the basic unit of resource on cloud data center using 0-1ILP with RTT equals 150 ms. As can be seen from the figure, the partitioned results present a clustered feature, which means that the method that would executed on the cloud data center reside very close to each other. Similar results are observed with different RTTs and applications [89]. The clustered code partition layout is consistent with our intuition that clustered execution would avoid extra transferring cost. Moreover, since the execution cost on the cloud is less compared to the execution cost on the mobile device, sequentially offloading further saves the cost by reducing the execution cost.

In the following part of this subsection, we would proof two features of the minimal offload property, which are

1. Once a method is decided to be run on the cloud computing infrastructure, all its descendants methods should be running on the cloud data center.

2. If a method is unremoteable or decided not to be run on the remote side, but at least two of its descendants are decided to be run on the cloud computing infrastructure, all the remoteable methods that are between the two methods can be offloaded.

The proof of the first part is as follows:

Proof. In mobile cloud computing, the computation capacity of the mobile device is lower

compared to the cloud computing server, and this is still true when the basic unit of resource is allocated on the server side. Hence the execution time of the same method would be less when it runs on the cloud computing than on the mobile device. Let's consider a simple example with three nodes as illustrated in Fig 6.4, where all the three methods can be executed both locally and remotely. Assume that the caller of method A will be run on the cloud data center and T_{A1}, T_{B1}, T_{C1} are the time of transferring the input state, T_{A2}, T_{B2}, T_{C2} are the time of transferring the return state, E_{A1}, E_{B1}, E_{C1} are the time of executing the method on the mobile device, while E_{A2}, E_{B2}, E_{C2} are the time of executing the method on the cloud data center using the basic resource unit. Since the computation capacity of the mobile device is lower compared withe the cloud data center, we have $E_{A1} > E_{A2}, E_{B1} > E_{B2}, E_{C1} > E_{C2}$. Hence, if method A is to be run on the server, then in the minimal overall time scheme, method B and C would also be run on the server, since it eliminates the time of transferring the input and returns state as well as saves time by running the method on the server. □

Another feature of the minimal offload property is that if a method and at least two of its descendants are to be run on the server side, all the methods between these two descendants should be run on the server. The proof is as follows:

Proof. Let's consider a general case as shown in fig. 6.5 where all the shown methods are remoteable. Assume method A is decided not to be run on the server side and two of its descendants method B and method D are decided to be run on the server side. Since within a single thread, all the methods are run sequentially, which means for any method to run, all its left-hand side brothers are already run. Our claim is that to minimize the overall time any method C_i should be run on the server side. We use the contradiction methodology to prove our point. Consider a partition P_k which at least one method C_k is decided to run on the mobile device. Compared to the partition where each C_i to be run on the server, P_k would definitely takes more time since it needs both more transferring time and execution time. □

6.4.1.2 Minimal Overall Time

In this part, we describe how to find the partition with minimal overall time leveraging the minimal download property we just proved. As shown by the minimal download property,

1. Once a method is decided to be run on the cloud data center, all its descendants methods should be running on the cloud data center.

2. Once a method is not decided to be run on the cloud computing infrastructure, its descendants should also be run on the server in a clustered manner.

We define the minimal remote couple of each method is the starting and ending point of two methods that would locally minimize the overall time if the method itself is decided not to be run on the server side. The feasible searching space of each method either its minimal remote couple or itself to be run on the server. In this way, instead of the $O(2^V)$ feasible searching space of the 0-1 ILP, the searching space now is $O(V)$, since when a method is decided not to be run on the server, the searching space would be reduced to the minimal remote couple.

6.4.1.3 Resource Estimation

Leveraging the minimal offload property, a slow start and binary search algorithm to estimate the resource demand of the mobile user is provided. The approach we adopt here is similar to the slow-start algorithm in TCP congestion-avoidance [28]. The idea can be explained as follows, at first, we allocate the resource on the server side using the basic unit of resource and get the minimal overall time. If this overall time satisfies the user SLA requirement, the corresponding partition would be the one and the resource demand would be the basic resource unit. Else if this overall time exceeds the user specified SLA, double the resource allocated on the server side. Repeat this doubling procedure until the overall time satisfies the user specified SLA. Then use binary search to find the minimal resource demand within a range of ε.

The pseudocode code in Algorithm 9 shows how to conservatively find the minimal resource demand that satisfies the SLA requirements. The first while loop finds a feasible partition and its corresponding resource demand that satisfy the user SLA requirement. The second while loop finds the minimal resource demand and its corresponding partition that satisfy the user SLA requirement.

6.4.1.4 Resource Estimation for Budget Limit Jobs

Until now we have illustrated how to obtain the resource demand and its corresponding partition of the completion-time aware jobs. In this following part, we will describe how to leverage the proposed method to obtain the resource demand of the budget limit jobs in Algorithm 10. Similarly to the resource estimation algorithm of the completion time-aware jobs, this algorithm finds the resource demand that is closest to its budget limitation.

Algorithm 9 Slow Start Binary Search Resource Estimation

Symbols: A – the particular application or software in discussion, D – user defined completion time, ε – the acceptable tolerance with regard to the user defined completion time, r – the amount of resource allocated to that user on the cloud computing infrastructure, r_0 – the basic unit of resource on the cloudd computing infrastructure, r_1 – the last explored amount of resource on the cloud computing infrastructure, t – the minimal overall time with a specified allocated resource
$r \leftarrow r_0$
$t \leftarrow code_partition(A, r_0)$
if $t \leq D$ **then return** r_0
end if
$r \leftarrow 2 * r$
$r_1 \leftarrow r_0$
while $t \geq D$ **do**
    $t \leftarrow code_partition(A, r)$
    $r_1 \leftarrow r$
    $r \leftarrow 2 * r$
end while
while $|D - t| \geq \varepsilon$ **do**
    $r \leftarrow \frac{r+r_1}{2}$
    $t \leftarrow code_partition(A, r)$
end while

Algorithm 10 Resource Estimation for Budget Limit Jobs

Symbols: A – the particular application or software in discussion, B – user defined budget limit, G – the call graph of A, V – the set of methods in A, E – the set of edges of the corresponding call graph of A, r_1 – the last explored amount of resource on the cloud data center, $w(e)$ – the cost of edge e, $g(v)$ – the cost of method v, $resource\_estimation\_time(G, D)$ – the resource estimation algorithm of call graph G with completion time of D, and
for each e in E **do**
    $w(e) \leftarrow 0$
    $w(v) \leftarrow g(v)$
end forreturn $resource\_estimation\_time(G, B)$

6.5 Evaluation

In this section, we first evaluate the code partition and resource estimation algorithm. In order to better understanding our method, this work implements the remote execution platform on Android devices and a server. As illustrated in the following, the proposed partition and estimation algorithm can accurately render a partition result given different user SLA.

Designing an accurate and efficient code partition algorithm for cognition augmenting applications is challenging. This kind of applications is usually sophisticated and contains a number of remoteable components [10] (i.e., methods that can be offloaded to the cloud). To better understand the characteristics of such applications, we downloaded 3 popular apps (Adobe Photoshop Express [66], Gesture Search [67], and Visidon AppLock [68]) from Google Play. We use reverse engineering tools [51, 52] to disassemble the dex files and find that each of them contains more than 20 remoteable components. We also built two applications one face recognition application (The face recognition application is built upon an open source code, which implements the Eigenface face recognition algorithm [43]), and one natural language processing application (The natural language processing application is built upon an open source code [44]) from open source code, and find that each of them contains also more than 15 remoteable components. The time required for solving the code partition problem by existing works with a 0-1 Integer Linear Programming (0-1 ILP) solver [10,23] is not negligible. For example, in the natural language processing application with 73 components the time needed for partition with 0-1 ILP solver is 7.3s compared to 21s time saving.

After getting the resource demand information, the joint code partition and resource allocation problem becomes a pure cluster resource scheduling problem and is solved by our [88] work. Here we just present several results from that work.

6.5.1 Code Partition and Resource Estimation

We implement our code partition and resource estimation algorithm in C on a dual core Intel Pentium 2.0GHz processor, 4G RAM desktop. Even though the code partition method itself is totally different from [89], the input of the code partition algorithm is the same, which is a call graph which represents the call graph with the following fields: execution time on the desktop, execution time on the mobile device, a pointer to its first child, a pointer to its next sibling, size of the input state to invoke this method, size of the return state, and the network bandwidth. The output of the two algorithms is the partition layout. To obtain the execution time of a method, we insert a timer in each method and log the execution time both on the mobile device and the desktop. The smartphone being used here is the Samsung

Galaxy ACE GT-S5830 with 278 MB RAM and Android 2.3.3 OS. The desktop is equipped with a dual core Intel Pentium 2.0GHz processor and a 4G RAM. To get the data size, we currently use the Sizeof tool developed by Roubtsov [32]. The Sizeof tool calculates an object's size in the heap, which is sufficient for obtaining the input of a partition algorithm. In real implementation scenario, we need to actually access the object in heap. This could be done via the reflection attribute of the Java language.

As the same with [89], we simulate the transfer time of the input and return state using the following rough estimation:

$$T = \frac{state \quad size}{bandwidth} \tag{6.5.1}$$

6.5.1.1 Partition Accuracy

As proved in section 6.4, the proposed code partition algorithm is an optimal polynomial time algorithm that would find the partition with minimal overall time. The resource estimation algorithm is also an optimal solution that is bound to find the resource demand closest to user's SLA requirement within $O(V^2 \log_2 V)$ time. Hence, the partition accuracy is evaluated against the heuristic algorithm in our [89] work.

Fig. 6.6 shows the comparison of the optimal minimal time partition with call link based partition in terms of overall time. Since the proposed scheme in this paper is an optimal solution, hence the partition performance is better compared with the heuristic solution in [89]. Note that the gab between the optimal solution in this paper and the heuristic solution gets larger when network delay becomes longer, which shows that the performance of the heuristic algorithm gets worse when network condition gets worse. The optimal solution enlarges the possibility of using cloud data center, which is the primary gain of mobile cloud computing.

6.5.1.2 Partition Efficiency

In this part, we will evaluate the code partition speed of the proposed optimal code partition algorithm compared with both 0-1 ILP [9, 10] and call link partition [89]. The result is shown in Table 6.1.

As can be seen from the table, the partition speed of the proposed scheme is much higher than the brutal force 0-1 ILP solver and is quite acceptable with the fact that the computation capacity of the cloud data center is high.

Table 6.1: Code Partition Efficiency.

Time (ms)	MinDownload	Brutal Force 0-1 ILP	CallLink
FaceReg	12.4756	886.9383	4.5051
NLP	70.2601	7282.8045	19.4476

6.5.1.3 Resource Estimation

In this part, we evaluate our resource estimation algorithm. As far as we are aware, there is no previous work on the resource estimation in mobile remote execution scenario and our algorithm is an optimal one, hence, no literature is available to be compared with. In this case, the resource estimation evaluated with different SLA requirements.

As can be seen from Fig. 6.7, the portion of methods run locally grows with the user specified completion time. This is because our resource estimation algorithm always conservatively require the minimal amount of the resource on the cloud data center, hence, when the SLA requirements releases, more methods would run on the mobile side.

Fig. 6.8 shows the portion of methods run on the server with regard to the budget limit. Here we assume renting the basic unit for one minute cost 100. Contrast to the completion time-aware jobs, the budget limit jobs tend to use more server resource as possible within their user specified budget. Hence, it shows a different trends from the the completion time-aware jobs.

6.5.2 Overall Performance

We evaluate the overall performance of the joint code partition and resource scheduling algorithm in terms of the total number of supported jobs in the system compared to the work in [89]. Since we don't have the real workload trace with call graph information, we synthesize the mobile workload by ourselves. The workload generation procedure is as follows:

1. we first generate the total number of methods in each job. The number of methods is drawn from uniform distribution with the range [10, 100].

2. the weight of each method, which is the resource demand of the method when the basic resource unit is allocated is drawn from uniform distribution with the range [0.01, 1].

3. the size of input and output states are drawn from uniform distribution with the range

[100, 1000].

We vary the computation capacity on the server side, hence the allocated resource using [89] would also change correspondingly. The result is shown in Fig. 6.9.

In Fig. 6.9, the server capacity is normalized to the basic resource unit while the Y axis is the job completion rate. As can be seen from the figure, using the joint code partition and resource scheduling algorithm achieve higher job completion rate than the work in [89]. We also note that the completion rate using [89] would first increase with the server capacity and then drop. This can be explained by the fact that when more resource are allocated to a single job, the utilization ratio of the cluster would be low, hence less jobs would be accommodated.

6.6 Related Work

6.6.1 Mobile Code Partition

Several works are proposed to design systems that combine the computation efforts of multiple machines. Agent Tcl [8] was one such a system that allowed a programmer to easily let computation jump from one endpoint to another. MAUI [10] enabled automated offloading and demonstrated that offloading could be an effective tool for performance and energy gains. However, it still requires annotation of what could be offloaded by developers. CloneCloud [9] to extend this design by using static analysis to automatically decide what could be offloaded.

The code partition problem for mobile cloud computing can be categorized by the granularity and partition algorithm been used, current work follows three approaches.

6.6.1.1 Full migration

[35, 36] can also be viewed as an example of the software as a service, which is often adopted by early stage research. The inflexibility and coarse granularity of full migration are the main drawback of this approach.

6.6.1.2 Pre-calculated offline partitions

Pre-calculated offline partitions CloneCloud [9] follows this approach. It aims to profile as many execution paths as possible by alternating the inputs and partitions each execution path under two different network conditions (WiFi and 3G) for time metric or eight states (<CPU, Scr, Net> triples) for energy metric. The smartphone searches for a matching partition in the database at run time. However, real network and device conditions cannot be generalized into fixed amount of states, and using the pre- calculated partitions cannot cover all the offloading scenarios.

6.6.1.3 Making decision at runtime

Examples of this approach are Odessa [22], MAUI [10] and Wishbone [23]. Odessa uses a greedy strategy by estimating whether to offload the current stage would be beneficial. Although the greedy algorithm is fast and can adapt to input and network changes, the decision is quite unreliable. MAUI and Wishbone formulate the partition problem into a 0-1 ILP where each variable is an indicator of whether the corresponding method should be offloaded. By using the previous profile of last run as an estimator of current run, the whole call graph can be partitioned with regard to the current network condition. However, the cost of solving the 0-1 ILP is not negligible in terms of time and memory even if it is on the server side, which is a NP-hard problem and no polynomial solution is available.

The most recent work of mobile cloud resource scheduling are [81, 82]. [81] proposes the idea to use computation offloading as a service to mobile devices in order to jointly solve the code partition and resource sharing problem. The problem with this work is that it doesn't consider the scheduling time cost. However, as it is shown in the evaluation part of this work, the scheduling actually is critical to the overall performance. [82] presents a mobile cloud offloading system that considers the end-to-end delay of cloud-enhanced mobile applications. The problem is that it assumes infinite resource on the cloud side. [95] models the code partition and offloading problem into the minimum cut problem and propose an polynomial time k-cut approximation algorithm. The problem with their work is that there exist some unremoteable components which cannot be represented by the the minimum cut model

6.6.2 Cluster Resource Scheduling

A number of works have been proposed to allocate resource among a fixed number of machines. Yanagisawa et al. [94] presented a resource allocation approach that assigns virtue

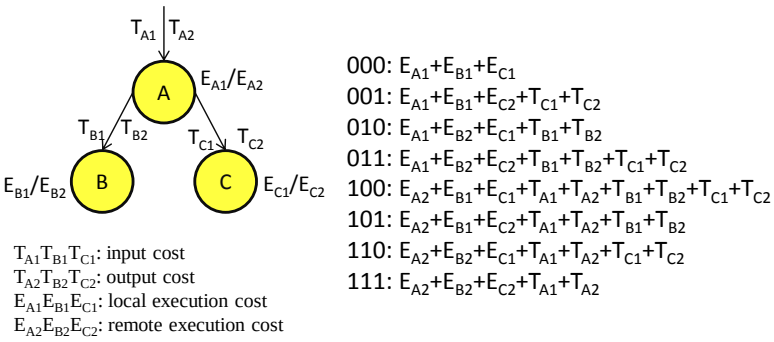
machines to physical machines based on the fluctuated CPU demand using mixed integer programming. To deal with the long processing time of the mixed integer programming solver, they developed a technique to partition the time-horizon into intervals and allocate the resource periodically. The division of intervals makes the scheme insufficient for an online solution. Niu et al. [34] proposed an iterative resource allocation algorithm with the goal of maximizing the sum of user utilities minus the cloud operator cost. The convergence time of their approach is 5 times faster than the gradient methods. Nevertheless, as illustrated in [24], the speed improvement though significant, still cannot fill up the gap towards an online scheduling. Jalaparti et al. [71] designed an online scheduling scheme to allow the cloud operators to choose among all the eligible resource combination to maximize the utilization ratio for MapReduce jobs. Wieder et al. [85] developed a system to select the most appropriate VM for the uses for MapReduce computations, so that users can specify their goals, such as minimizing monetary cost or completion time. The problem of [71, 85] lies in that they still assume the fixed and pre-defined VM configurations. As a result, the resource utilization in these approaches is still limited. Our approach differs from these existing works in that we exploit the distinct resource requirements for two different types of jobs and make a tradeoff between them, while able to schedule resources in an online and fairly accurate manner. In order to achieve the job specified SLA and to efficiently use the resources in the cloud data center, effective resource scheduling mechanism should be employed to carefully assign the incoming jobs onto the appropriate physical machines. Several studies have been proposed to improve the resource efficiency in public cloud data center. To handle heterogeneous workloads, Bhattacharya et al. proposed [54] Hierarchical Dominant Resource Fairness (H-DRF), which is an online scheduler that achieve high efficiency scheduling by equalizing the dominant resource share between each pair of sibling nodes in the hierarchy. [2] proposed an estimation-based opportunistic scheduling algorithm that can dynamically allocate the resources to each job while maintaining high usage ratio. However, none of these approaches could be directly applied to the mobile cloud computing environment. The problem lies in the interaction between the mobile code partition and the resource scheduling. The combination of mobile code partition and workload diversity brings new challenges that do not exist in the traditional resource scheduling setting, and are not fully addressed by the above-mentioned efforts.

6.7 Chapter Summary

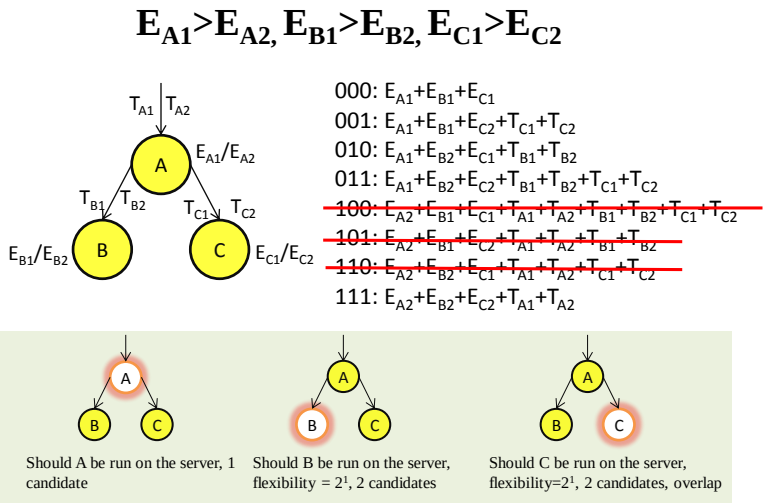
Mobile cloud computing is a new computing paradigm that enables mobile users to demand computing services from ubiquitous computing infrastructures to access various kinds of applications, softwares, and even operation systems. In order to achieve the job specified SLA and to efficiently use the resources in the cloud computing infrastructure, effective resource scheduling mechanism should be employed to carefully assign the incoming jobs

onto the appropriate physical machines. However, no existing approaches could be directly applied to the mobile cloud computing environment. The problem lies in the interaction between the mobile code partition and the resource scheduling. The combination of mobile code partition and workload diversity brings new challenges that do not exist in the traditional resource scheduling setting, and are not fully addressed by the above-mentioned efforts.

In this paper, we propose a reservation-based code partition and resource scheduling algorithm, which is a joint code partition and resource scheduling algorithm. We first estimate the reservation threshold of the delay-sensitive or completion time-aware jobs, and conservatively estimate resource that is available on the cloud computing infrastructure. Giving the estimated available resource, an linear time code partition algorithm is proposed to get the minimal completion time. An adjustment of the allocated resource is made for delay-sensitive or completion time-aware jobs if the minimal completion time violates the user specified SLA.



(a) A general graph.



(b) In mobile cloud computing.

Figure 6.4: First feature of minimal offload property: once a node is to be run on the server side, all its descendants should be run on the server.

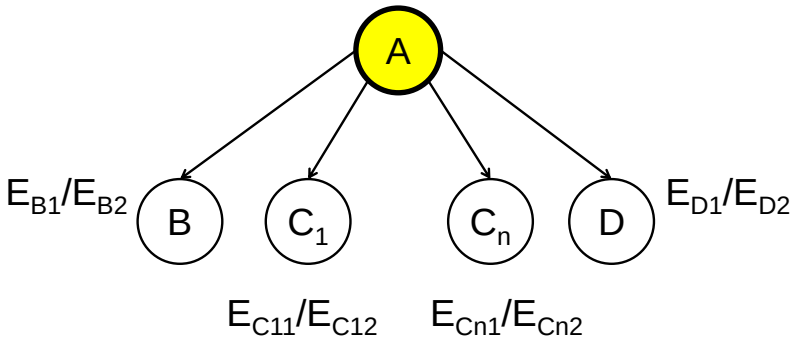
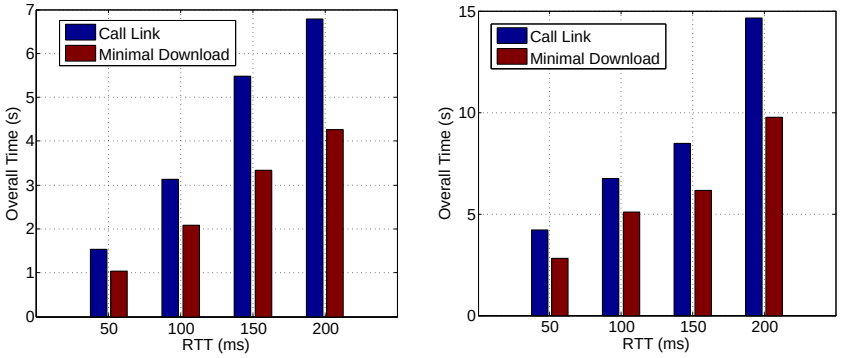


Figure 6.5: Second feature of minimal offload property: once a node is to be run on the mobile side, its descendants that would run on the server are clustered.



(a) Face Recognition.

(b) Natural Language Processing.

Figure 6.6: Comparison of the optimal minimal time partition with call link based partition in terms of overall time.

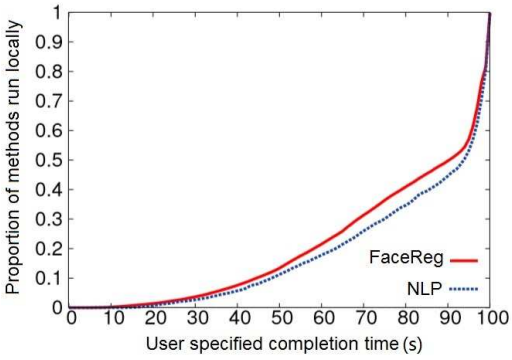


Figure 6.7: Portion of methods run locally vs. user specified completion time.

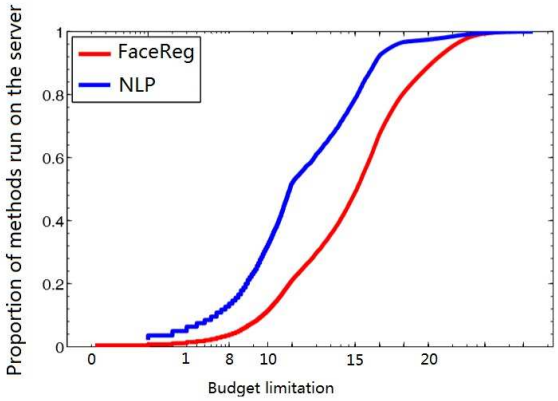


Figure 6.8: Portion of methods run on the server vs. user specified budget limit. Assuming rent the basic unit for one minute be 100.

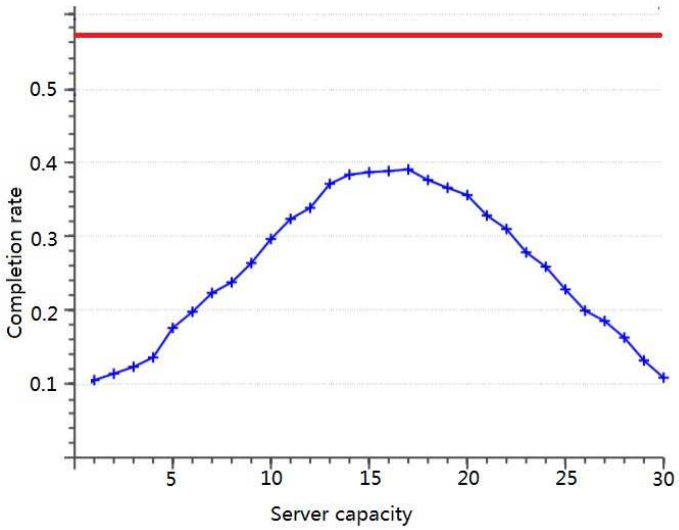


Figure 6.9: Completion rate of the jobs with varying server capacity. Red line: minimal feasible resource demand; blue line: minimal overall time partition in CloudNet.

Chapter 7

Dissertation Conclusion

This work argues that using finer-grained resource scheduling can benefit both the cloud operator and the tenants. In addition to evaluation metrics of job completion rate and resource utilization rate, we also consider the overall revenue of the cloud operator as the optimization objective which can be seen as a direct reflection of the cloud operator's interest. This also results in meeting the job's deadline. With objective of maximizing the revenue and taking into account the data center's capacity constraint, the multi-resource scheduling problem is modeled as an integer programming problem. However, the traditional integer programming solvers are not applicable here due to the strict requirement for online resource scheduling to determine the solution quickly and schedule the job so that it completes within the deadline.

Using finer-grained resource scheduling can benefit both the cloud operator and the tenants. In addition to evaluation metrics of job completion rate and resource utilization rate, we also consider the overall revenue of the cloud operator as the optimization objective. This also results in meeting the job's deadline. With objective of maximizing the revenue and taking into account the data center's capacity constraint, the multi-resource scheduling problem is modeled as an integer programming problem. However, the traditional integer programming solvers are not applicable here due to the strict requirement for online resource scheduling to determine the solution quickly and schedule the job so that it completes within the deadline. As summarized in [24], even for a very small scale of jobs and number of servers, the time to solve the problem would be unacceptable. For example, in one case tested with 4 machines and 22 jobs, the solution time is 1957.6 seconds on Intel Xeon E5420 2.50 GHz computers with 6 MB cache and 6 GB memory.

We analyze the resource demands of jobs from a public cluster trace [49], and find out that they have a clear relationship to the job duration. The relatively small number of long duration jobs consume the majority of the computing resources. This motivates our approach to tradeoff between scheduling accuracy and scheduling speed: since the few long

duration jobs consume most of the resources, a sophisticated scheduling algorithm that can reach an optimal solution is used; on the other hand, for short duration jobs, due to their large number, fast scheduling is important. Therefore, we propose a combined constraint programming and heuristic scheduling algorithm. We use constraint programming as the tool to find an optimal scheduling for the long duration jobs. For the jobs with short durations, our experiments show that simple heuristics such as first fit or best fit suffice.

Recent studies like [58,61,70,74,79] show that the applications run on cloud have diverse service level requirements in terms of throughput, latency, and jitter. These applications can be coarsely classified into two types: deadline-sensitive applications and best-effort applications [61, 70, 80]. The deadline-sensitive applications often consume large amounts of resource and have long durations. Furthermore, they are often critical to business, and have strict deadline requirements. The best-effort applications typically consume less resource and do not have explicit deadlines. However, finishing the job at an earlier time would improve the users' satisfaction. Based on this observation, we believe that it is highly desirable to design a cloud scheduling system aiming at guaranteeing the deadline requirements of the deadline-sensitive applications as well as to improve the completion time of the best-effort applications. Through extensive trace-driven analysis, we find that the number and the resource consumption of the deadline-sensitive and best-effort applications follows the 80/20 principle: While a majority of these applications are best-effort applications, up to 80% of the cloud data center resources are consumed by deadline-sensitive applications [61,74,79]. We will leverage this property to design an efficient resource scheduling algorithm later.

In this paper, we show that the multiple Pareto optimal VM set property of the applications can be used to improve the overall utilization of the system. We propose a multi-resource scheduler that allocates applications to VM sets and then to physical machines with the goal to minimize the number of activated physical machines as well as to maximize the overall utility. The utility function of the deadline-sensitive applications is defined as a 1-0 function where the utility would stay the same when it is within its deadline and drop to zero when the it exceeds its deadline. For the best-effort applications, a soft utility function is used where the utility drops gradually [70]. We first provide a greedy yet optimal algorithm to find the VM set that can minimize the activated physical machines and still satisfy the resource requirement. Using the VM sets as input, we model the VM scheduling problem as an integer programming problem with the goal to improve the overall utility of all applications. By leveraging the 80/20 property of the deadline-sensitive applications and best-effort applications, we propose a combined constraint programming and heuristic algorithm. We reserve a dynamic number of physical machines for the deadline-sensitive applications and use constraint programming to get a more accurate scheduling result. Due to the huge number and the less resource consumption of the best-effort applications, we use heuristic scheduling policy to get a fast yet acceptable scheduling result. Trace-driven simulation shows over 25% improvement of the overall utility compared to best-fit and first-

fit using the optimal VM selection algorithm, over 30% improvement compared to best-fit and first-fit using random VM selection scheme and 18% improvement compared to the Bazaar-I approach.

Chapter 8

Bibliography

Bibliography

[1] A. A. Bhattacharya, D. Culler, E. Friedman, A. Ghodsi, S. Shenker, and I. Stoica, Hierarchical Scheduling for Diverse Datacenter Workloads, SoCC, 2013.

[2] E. Boutin, J. Ekanayake, W. Lin, B. Shi, J. Zhou, Z. Qian, M. Wu, and L. Zhou, Apollo: Scalable and Coordinated Scheduling for Cloud-Scale Computing, OSDI, 2014.

[3] C. J. Wong, S. Sen, T. Lan, and M. Chiang. Multi-resourceallocation: Fairness-efficiency tradeoffs in a unifying framework. In INFOCOM, pages 1206–1214, 2012.

[4] A. Ghodsi, V. Sekar, M. Zaharia, and I. Stoica. Multi-resource fair queueing for packet processing. In SIGCOMM, 2012.

[5] H. Ying, W. Yang, G. Wang, Wireless Multicast for Mobile Transparent Computing, High Performance Computing and Communications (HPCC), 2013.

[6] W. Liang, Y. Xiong, M. Wu, A Cross Platform Computing Method and Its Application for Mobile Device in Transparent Computing, High Performance Computing and Communications (HPCC), 2013.

[7] H. Liu. A Measurement Study of Server Utilization in Public Clouds. Int. Conf. on Cloud and Green Computing (CGC), Dec. 2011.

[8] R. S. Gray, Agent Tcl: A flexible and secure mobile-agent system, Technical Report, Dartmouth College Hanover, 1998

[9] B. G. Chun, S. Ihm, P. Maniatis, M. Naik and A. Patti. Cloud: Elastic Execution between Mobile Device and Cloud. EuroSys 2011.

[10] E. Cuervo, A. Balasubramanian, D. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, MAUI: Making Smartphones Last Longer with Code Offload, MobiSys 2010.

[11] C. Wang, T. Javidi and George Porter. End-to-End Scheduling for All-Optical Data Centers. INFOCOM 2015.

[12] G. Birkhoff. Tres observaciones sobre el algebra lineal. Univ. Nac. Tucuman Rev. Ser.

A5, no. 147-150, 1946.

[13] L. Wang, A. Nappa, J. Caballero, T. Ristenpart and A Akella. WhoWas: A Platform for Measuring Web Deployments on IaaS Clouds. INFOCOM IMC 2014.

[14] M. Conley, A. Vahdat and G. Porter. Achieving Cost-efficient, Data-intensive Computing in the Cloud. ACM SoCC 2015.

[15] E. Walker. Benchmarking Amazon EC2 for high-performance scientific computing. LOGIN, 2008.

[16] P. Mehrotra, J. Djomehri, S. Heistand, R. Hood, H. Jin, A. Lazanoff, S. Saini, and R. Biswas. Performance evaluation of Amazon EC2 for NASA HPC applications. In ScienceCloud, 2012.

[17] G. Wang and T. E. Ng. The impact of virtualization on network performance of Amazon EC2 data center. IEEE INFOCOM, 2010.

[18] J. Schad, J. Dittrich, and J. Quiane-Ruiz. Runtime measurements in the cloud: Observing, analyzing, and reducing variance. In VLDB, 2010.

[19] F. Chen, M. Kodialam, and T. Lakshman. Joint scheduling of processing and shuffle phases in mapreduce systems. IEEE INFOCOM, 2012.

[20] http://www.gecode.org/doc-latest/reference

[21] C. Delimitrou, D. Sanchez, and C. Kozyrakis, Tarcil: Reconciling Scheduling Speed and Quality in Large Shared Clusters, SoCC 2015.

[22] M. R. Ra, A. Sheth, L. Mummert, P. Pillai, D. Wetherall, and R. Govindan. Odessa: Enabling Interactive Perception Applications on Mobile Devices. MobiSys 2011.

[23] R. Newton, S. Toledo, L. Girod, H. Balakrishnan, and S. Madden. Wishbone: Profile-based Partitioning for Sensornet Applications. NSDI 2009.

[24] S. Heinz, J. C. Beck. Solving Resource Allocation/Scheduling Problems with Constraint Integer Programming. Workshop on Constraint Satisfication Techniques for Planning and Scheduling Problems, 2011.

[25] M. Mihailescu, Y. M. Teo. Dynamic Resource Pricing on Federated Clouds. CCGrid 2010.

[26] L. Popa, G. Kumar, *et al.* FairCloud: Sharing the Network in Cloud Computing. ACM SIGCOMM 2012.

[27] H. Roh, C. Jung, *et al.* Resource Pricing Game in Geo-distributed Clouds. IEEE INFOCOM 2013.

[28] W. Stevens, RFC2001: TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms, tools.ietf.org/html/rfc2001

[29] H. Zhang, B. Li, H. B. Jiang. A Framework for Truthful Online Auctions in Cloud Computing with Heterogeneous User Demands. IEEE INFOCOM 2013.

[30] Y Zheng, N. B. Shroff, R. Srikant and P. Sinha. Exploiting Large System Dynamics for Designing Simple Data Center Schedulers. IEEE INFOCOM 2015.

[31] Y. Chen, S. Alspaugh, A. Ganapathi, R. Griffith, R. Katz, Statistical Workload Injector for MapReduce(SWIM): https://github.com/SWIMProjectUCB/SWIM/wiki

[32] V. Roubtsov, Sizeof, http://www.javaworld.com/javaworld/ javatips/jw-javatip130.html

[33] M.P.J. Fromherz. Constraint-based scheduling. American Control Conference, 2001.

[34] D. Niu, B. Li. An Efficient Distributed Algorithm for Resource Allocation in Large-Scale Coupled Systems. IEEE INFOCOM 2013.

[35] S. Osman, D. Subhraveti, G. Su, and J. Nieh. The Design and Implementation of Zap: A System for Migrating Computing Environments. OSDI 2002.

[36] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies. The Case for VM-based Cloudlets in Mobile Computing. IEEE Pervasive Computing, 8(4), 2009.

[37] A. Wieder, P. Bhatotia, *et al.* Orchestrating the Deployment of Computations in the Cloud with Conductor. USENIX NSDI 2012.

[38] L. E. Dubois, G. Mounie, D. Trystram, Analysis of Scheduling Algorithms with Reservations. IEEE IPDPS 2007.

[39] http://aws.amazon.com/solutions/case-studies/netflix/

[40] http://aws.amazon.com/solutions/case-studies/notre-dame/

[41] http://aws.amazon.com/solutions/case-studies/major-league-baseball/

[42] P. Garraghan, P. Townend, J. Xu. An Analysis of the Server Characteristics and Resource Utilization in Google Cloud. IEEE Int. Conf. on Cloud Engineering, Sep. 2013.

[43] http://darnok.org/programming/face-recognition/

[44] http://nlp.stanford.edu/software/index.shtml

[45] http://aws.amazon.com/ec2

[46] http://aws.amazon.com/ec2/instance-types/

[47] https://aws.amazon.com/ec2/purchasing-options/

[48] Google Compute Engine Pricing. https://cloud.google.com/products/compute-engine/

[49] https://code.google.com/p/googleclusterdata/

[50] http://www.gecode.org/

[51] http://code.google.com/p/dex2jar

[52] http://java.decompiler.free.fr/?q=jdgui

[53] Q. Zhang, M. Zhani, R. Boutaba, and J. L. Hellerstein. Dynamic Heterogeneity-aware Resource Provisioning in the Cloud. ICDCS 2013.

[54] A. Bhattacharya, D. Culler, E. Friedman, A. Ghodsi, S. Shenker, and I. Stoica. Hierarchical Scheduling for Diverse Datacenter Workloads. ACM SOCC 2013

[55] N. Bansal, A. Caprara, M. Sviridenko. A New Approximation Method for Set Covering Problems, with Applications to Multidimensional Bin Packing. SIAM J. Comput., 2009, Vol.39, No.4, pp.1256-1278.

[56] R. Bartak. Guide to Constraint Programming. http://kti.ms.mff.cuni.cz/∼bartak/constraints/intro.html

[57] N. Beldiceanu and M. Carlsson. A New Multi-Resource Cumulatives Constraint with Negative Heights. Principles and Practice of Constraint Programming, 2002.

[58] R. Boutaba, L. Cheng and Q. Zhang. On cloud computational models and the heterogeneity challenge. J. Internet Services and Applications, 3(1):77–86, 2012

[59] C. Reiss and J. Wilkes, Google cluster-usage traces: format + schema, 2011

[60] L. Zhang, C. Wu, Z. Li, C. Guo, M. Chen and F. C.M. Lau. Moving Big Data to The Cloud: An Online Cost-Minimizing Approach. IEEE Journal on Selected Areas in Communications (JSAC), 2015.

[61] C. Curino, D. Difallah, C. Douglas, S. Krishnan, R. Ramakrishnan and S. Rao. Reservation-based Scheduling: If You're Late Don't Blame Us! ACM SOCC 14.

[62] D. Gamarnik. Stochastic Bandwidth Packing Process: Stability Conditions via Lyapunov Function Technique. Queueing Systems, 2004, Vol.48, pp.339-363.

[63] J. Cohen, Embedded speech recognition applications in mobile phones: Status, trends, and challenges, IEEE ICASSP 2008.

[64] Apple Litigation Marks A New Era Of Expectations For Voice Recognition, http://seekingalpha.com/article/432161-apple-litigation-marks-a-new-era-of-expectations-for-voice-recognition, March 2012.

[65] J. C. Mogul and R. R. Kompella. Inferring the Network Latency Requirements of Cloud Tenants. In Proc. of USENIX HotOS, 2015

[66] Adobe Photoshop Express, https://play.google.com/store/apps/details?id=com.adobe.psmobile

[67] Gesture Search, https://play.google.com/store/apps/details?id=com.google.android.apps.gesturesearch

[68] Visidon AppLock, https://play.google.com/store/apps/details?id=visidon.AppLock

[69] G. Hua, Y. Fu, M. Turk, M. Pollefeys, and Z. Zhang, Introduction to the Special Issue on Mobile Vision, International Journal of Computer Vision, vol. 3, pp. 277-279 , 2012.

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

[71] V. Jalaparti, H. Ballani, P. Costa, T. Karagiannis, A. Rowstron. Bridging the Tenant-Provider Gap in Cloud Services. ACM SOCC 2012.

[72] J. W. Jiang, T. Lan, S. Ha, M. Chen, and M. Chiang. Joint VM Placement and Routing for Data Center Traffic Engineering. IEEE INFOCOM 2012.

[73] S. T. Maguluri, R. Srikant, and L. Ying. Stochastic Models of Load Balancing and Scheduling in Cloud Computing Clusters. IEEE INFOCOM 2012.

[74] A. Mishra, J. Hellerstein, W. Cirne, and C. Das. Towards characterizing cloud backend workloads: insights from Google compute clusters. ACM SIGMETRICS 2010.

[75] R. Panigrahy, K. Talwar, L. Uyeda, U. Wieder. Heuristics for Vector Bin Packing. In MSR TR, 2011.

[76] J. Xia, The third-generation-mobile (3G) policy and deployment in China: Current status, challenges, and prospects. Telecommunications Policy, 2011

[77] Global mobile statistics 2012 Home: all the latest stats on mobile Web, apps, marketing, advertising, subscribers and trends. http://mobithinking.com/mobile-marketing-tools/latest-mobile-stats/

[78] C. L. Pape. Constraint-Based Scheduling: A Tutorial.

[79] C. Reiss, A. Tumanov, G. Ganger, R. Katz, and M. Kozuch. Heterogeneity and Dynamicity of Clouds at Scale: Google Trace Analysis. ACM SOCC 2012

[80] M. Schwarzkopf, A. Konwinski, M. Malek, and J.Wilkes. Omega: flexible, scalable schedulers for large compute clusters. ACM EuroSys, 2013

[81] C. Shi, K. Habak, P. Pandurangan, M. Ammar, M. Naik, and E. Zegura. COSMOS: computation offloading as a service for mobile devices. ACM MobiHoc, 2014

[82] L. Ravindrantah, J. Padhye, R. Mahajan, and H. Balakrishnan. Timecard: Controlling User-Perceived Delays in Server-Based Mobile Applications. ACM SOCC 2014.

[83] A. L. Stolyar, Y. Zhong. A large-scale service system with packing constraints: Minimizing the number of occupied servers. ACM SIGMETRICS 2013, Pittsburgh, PA, Jun. 2013.

[84] M. Mihailescu, Y. M. Teo. Dynamic Resource Pricing on Federated Clouds. CCGrid 2010.

[85] A. Wieder, P. Bhatotia, A. Post, and R. Rodrigues, Orchestrating the Deployment of Computations in the Cloud with Conductor. USENIX NSDI 2012

[86] G. J. Woeginger. There Is No Asymptotic Ptas For Two-Dimensional Vector Packing. In Information Processing Letters, 1997.

[87] T. Park, J. Lee, I. Hwang, C. Yoo, L Nachman, and J. Song, E-Gesture: a collaborative architecture for energy-efficient gesture recognition with hand-worn sensor and mobile devices. SenSys 2011

[88] Y. Zhang, X. Fu, K. K. Ramakrishnan, Fine-Grained Multi-Resource Scheduling in Cloud Datacenters, The 20th IEEE International Workshop on Local and Metropolitan Area Networks (LANMAN 2014), invited paper, May 2014.

[89] Y. Zhang, H. Liu, L. Jiao, X. Fu, 1st IEEE International Conference on Cloud Networking (CloudNet 2012), Paris, France, November 2012.

[90] L. Zheng, C. J-Wong, C. W. Tan, M. Chiang, and X. Wang. How to Bid the Cloud. SIGCOMM 2015.

[91] F. R. Dogar, T. Karagiannis, H. Ballani, and A. Rowstron. Decentralized task-aware scheduling for data center networks. ACM SIGCOMM 2014.

[92] N. Jain, I. Menache, J. S. Naor, and Yaniv, J. Near-optimal scheduling mechanisms for deadline-sensitive jobs in large computing clusters. ACM Trans. on Parallel Computing, 2015.

[93] J. Nair, V. G. Subramanian, and A. Wierman. On competitive provisioning of cloud services. ACM SIGMETRICS 2014.

[94] H. Yanagisawa, T. Osogami, R. Raymond. Dependable Virtual Machine Allocation. IEEE INFOCOM 2013.

[95] Y. H. Kao, B. Krishnamachari, M. R. Ra and F. Bai. Hermes: Latency Optimal Task Assignment for Resource-constrained Mobile Computing. IEEE INFOCOM 2015.

YUAN ZHANG

Office Address: Georg-August-Universität Göttingen
Institute of Computer Science
Goldschmidtstr. 7
37077 Göttingen, Germany
Date of Birth: Dec. 14, 1985

Home Address: Bonhoefferweg 2
D-37075 Göttingen
Mobile Phone: +49-17661357319
or: +86-13269478826
Email: zhang@cs.uni-goettingen.de

Education

Sep. 1, 2011-Present **Ph.D. Candidate**, Computer Science, **University of Göttingen**

Sep. 1, 2008-Jun.29, 2011 **M.S.E.**, Computer Science, **Tsinghua University**, GPA 85.9, rank 38/219

Sep. 1, 2004-Jun. 20, 2008 **B.E.**, Telecommunications, **Beijing University of Posts and Telecommunications**, GPA 91.8, rank 1/585

Publications

1. **Yuan Zhang**, Xiaoming Fu, K. K. Ramakrishnan, *"Fine-Grained Multi-Resource Scheduling in Cloud Datacenters"*, 20th IEEE Workshop on Local and Metropolitan Area Networks (LANMAN 2014), May 2014.
2. **Yuan Zhang**, Hao Liu, Lei Jiao, Xiaoming Fu, *"To Offload or Not to Offload: An Efficient Code Partition Algorithm for Mobile Cloud Computing"*, 1st IEEE International Conference on Cloud Networking (CloudNet 2012), Nov 2012.
3. **Yuan Zhang**, Yongfeng Huang, Jian Yuan, Jiaxi Long, *"An efficient group rekey scheme for P2P IPTV DRM"*, IEEE Conference 2011, pp 1479-1483
4. **Yuan Zhang**, Yongfeng Huang, Bo Xiao, Baolin Liu, Detection of Steganographic VoIP Communications via Sliding Windows, Computer research and development, 2009.
5. Bin Xiang, Konglin Zhu, Xiaoyi Zhang, **Yuan Zhang**, Yumei Wang and Lin Zhang, *"Modeling and Crowdtesting Application-Based QoE in Mobile Networks"*, The 20th IEEE Symposium on Computers and Communications (ISCC), 2015.
6. Yongfeng Huang**, Yuan Zhang**, Shanyu Tang. *"Detection of Covert Voice-over Internet Protocol Communications Using Sliding Window-Based Steganalysis"*, IET Communications, 2012.
7. Chao Zhang, **Yuan Zhang**, Yongfeng Huang, Xing Li, Peer selection algorithm in mixed IPv4/6 P2P network, In proceeding of 2010 International Workshop on NGI and P2P Systems, 2010.7.

Research Projects

Mar. 20, 2013- Present **Resource Scheduling for Public Cloud**, University of Göttingen

1. Proposed a dynamic resource scheduling algorithm under heterogeneous workload scenario; the maximum revenue is bounded by Lyapunov optimization theory; this work is the first work by far that can take both job deadline and network topology into consideration.
2. Implemented a Hadoop based cloud platform, four benchmark applications are used to get real trace.
3. Experiment shows that the total revenue would improve by 40% compared to the static pricing model.

Sep. 1, 2011- Mar. 19, 2013 **Mobile Cloud Computing**, University of Göttingen

1. Designed and implemented an Android-based mobile cloud computing platform; using face recognition and natural language processing as the study case, an offloading middleware is developed which is also applicable for the Apps whose source code is not available.
2. Proposed a high speed code partition algorithm which can reduce the scheduling cost and save energy cost.
3. The code partition algorithm is test under 3G and WiFi environment, the overall execution time would increase by 30% compared to the Maui system.

May. 1, 2009- Jun. 29 2011 **Digital Right Management of P2P IPTV application (CNGI Program)**, Tsinghua University

1. Design and implemented a P2P IPTV DRM system and proposed an efficient key distribution algorithm.

2. The DRM module is deployed on the SIPP2P system and opened to students and employees in Tsinghua.
3. The maximum online number exceeds 200; shown as a demo platform at the 100 anniversary of Tsinghua.

Jan. 2008- Oct. 2008 **Information Hiding Detection in real time environment**, Tsinghua University

1. Designed and implemented the RS-based information hiding detection system.
2. The detection rate can reach 95% for last bit flip hiding.

Internships and Working Experience

Sep. 1, 2010-Jan. 31, 2011 **User Advertisement Projecting Algorithms Engineer,** Sogou Corporation

Sep. 1, 2009-Mar. 30, 2010 **Research Assistant,** China Telecom Beijing Research Institute

Selected Honors and Awards

Sep. 1, 2011-Aug. 30, 2015 CSC Scholarship, China Scholarship Council

Oct. 30, 2009 Honorable Scholarship, Tsinghua University

Jun. 20, 2008 Outstanding Bachelor Graduates, Beijing University of Posts and Telecommunications

Visiting Scholars

Sep. 1, 2014-Oct. 26, 2014 Visiting scholar at Beijing University of Posts and Telecommunications, Tsinghua University and Nanjing Unversity, sponsored by EU FP7 IRSES Mobile Cloud Project (Grant No. 612212)

Aug. 26, 2012-Sep. 8, 2012 Visiting scholar at Fudan University, sponsored by DAAD/CSC PPP China Project Principles and Design of Next-Generation Internet Media Streaming

Teaching Experiences

Oct. 1, 2014-Mar 30, 2015, Practical Course Networking Lab, Teaching Assistant, University of Göttingen

Apr. 1, 2014-Sep. 30, 2014, Seminar on Internet Technologies, Teaching Assistant, University of Göttingen

Oct. 1, 2012-Mar 30, 2013, Advanced Topics in Computer Networking, Teaching Assistant, University of Göttingen

Apr. 1, 2012-Sep. 30, 2012, Advanced Topics in Mobile Communications, Teaching Assistant, University of Göttingen

Professional Activities

Paper reviews on IEEE International Conference on Computer Communications (INFOCOM)

IEEE International Conference on Network Protocols (ICNP)

IEEE/ACM International Symposium on Quality of Service (IWQoS)

EURASIP Journal on Wireless Communications and Networking (JWCN)

Computer Communications

References

Professor Xiaoming Fu Institute of Computer Science, Georg-August-Universität Göttingen

Professor K. K. Ramakrishnan Computer Science University of California Riverside

Professor Yongfeng Huang Electronic Engineering Department, Tsinghua University

Professor Lin Zhang Department of Telecommunications, Beijing University of Posts and Telecommunications

Professor Wenzhong Li Department of Computer Science and Technology, Nanjing University

Professor Yuezhi Zhou Department of Computer Science and Technology, Tsinghua University

Professor Jin Zhao School of Computer Science, Fudan University