A Cloud Eco-System: Reactive Demand Control and Dynamic Pricing Methodology

by

Yuanfang Chi

B.A.Sc., The University of British Columbia, 2012

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF APPLIED SCIENCE

in

The Faculty of Graduate and Postdoctoral Studies

(Electrical and Computer Engineering)

THE UNIVERSITY OF BRITISH COLUMBIA

(Vancouver)

October 2015

Abstract

Resources are limited in capacity. In the meanwhile, over-provisioning of resources resulted in server low utilization could be costly to cloud providers. The underlying reasons of the low utilization are multiple-folds, such as uneven application fit where the application cannot fully utilize the resources allocated or the uncertainty in demand forecasts that is introduced by the dramatically varied demand of the cloud resource between peak and non-peak periods. While many research works are devoted to optimize the resource allocation techniques in the effort of achieving higher server utilization, how to control resource demand so the correct level of resource provisioning can be determined has become the next research question. In this thesis, we introduce a pricing methodology with dynamic pricing that intended to induce desired demand pattern and enhances the revenue of a cloud provider. The proposed pricing methodology encourages cloud tenants, whose requested Virtual Machines (VMs) can be allocated easily, to use more cloud service by offering them lower prices and discouraging cloud tenants, whose requested VMs are difficult to allocate, from using cloud service by charging them higher prices. We study our pricing methodology with a combinatorial optimization algorithm, the Knapsack Algorithm and show that the overall revenue is enhanced through evaluations. Then, to achieve fairness among users, we further perform a case study of our pricing methodology with a multi-resource allocation fairness algorithm, the Heterogeneous Dominant Resource Fairness (DRFH) algorithm. Trace-driven simulation results show that the proposed pricing methodology with DRFH can increase the overall revenue by up to 11.60%. Furthermore, we propose a novel cloud federation system that is cognitive to the dynamic prices as a decision making assistant tool for our pricing methodology. The cloud federation system automatically selects and migrates user tasks to a cloud system that is charging at a more affordable rate. We discuss the architectural framework and platform design, provide a mathematical formulation and investigate a total service cost minimization approach with privacy constraints. Simulation results demonstrate the proposed system can lower the cost of cloud services by exploiting the advantages of dynamic prices of multiple clouds.

Preface

The pricing methodology proposed in this thesis is based on work conducted in UBC Wireless Networks and Mobile Systems Laboratory with Xiuhua Li, Dr. Xiaofei Wang, Professor Victor C.M. Leung and Professor Abdalalh Shami. As the first author of this paper, I initiated and conducted most of the research. I conducted all the simulations and wrote most of the manuscript. Xiuhua Li and Dr. Xiaofei Wang contribued in research ideas and mathematical models. Professor Victor C.M. Leung and Professor Abdalalh Shami helped in research directions and manuscript proof-reading. The following is the details of the published paper in the thesis.

The cognitive price-aware cloud federation system is based on work conducted in UBC Wireless Networks and Mobile Systems Laboratory with Wei Cai, Zhen Hong, Professor Henry C.B. Chan and Professor Victor C.M. Leung. As the first author of this paper, I contribued in research ideas, mathematical models and manuscript preparing. Wei Cai and Zhen Hong contribued in simulations and mathematical model refining. Professor Victor C.M. Leung and Professor Henry C.B. Chan helped in research directions and manuscript proof-reading. The following is the details of the published paper in the thesis.

- Yuanfang Chi, Xiuhua Li, Xiaofei Wang, Victor C.M. Leung and Abdalalh Shami, “A Fairness-Aware Pricing Methodology for Revenue Enhancement in Service Cloud Infrastructure”, accepted for publication in IEEE Systems Journal, 2015.

- Yuanfang Chi, Wei Cai, Zhen Hong, Henry C.B. Chan and Victor C.M.Leung, “A Privacy and Price-Aware Inter-cloud System”, to appear in 7th IEEE International Conference on Cloud Computing Technology and Science (CloudCom2015), Vancouver, Canada, Nov 30-Dec 3, 2015.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

Amazon EC2 Amazon Elastic Compute Cloud

CPU central processing unit

CSP Constraint Satisfaction Problem

DRF Dominant Resource Fairness

DRFH Heterogeneous Dominant Resource Fairness

FIFO first-in-first-out

GB gigabyte

IaaS Infrastructure as a Service

NP-hard non-deterministic polynomial-time hard

PM physical machine

PSM Price Sensitivity Measurement

QoS Quality of Service

RAID redundant array of independent disks

SLA Service Level Agreement

TRR Total Revenue Redistribution

TUPR Total Unit Price Redistribution

VBSE Value-Based Software Engineering

VM virtual machine

Notations

$.^T$ transpose

$||.||_1$ l_1 norm

α price elasticity coefficient

ω_d data access proportion

Θ maximum quantity of program codes allowed to be hosted in the same cloud

Φ maximum quantity of data sets allowed to be stored in the same cloud

cs number of available cloud service providers

c_l normalized resource capacity vector for each server l

dc number of data chunks

d_{ir} normalized resource vector of user i's Virtual Machine

$f_R(p)$ demand function

pc the number of program codes

w set of pricing weights

A matrix of resource allocation

B_d matrix of assignment of data chunks over cloud services

B_p matrix of assignment of program codes over cloud services

D_i resource vector of user i's demanded Virtual Machine

F_d set of data chunks

F_p set of program codes

$G_i(A_i)$ global dominant share allocated to user i

I matrix of network input volume

L matrix of code-data relationship

M matrix of message exchanges

$N_i(A_i)$ total number of VMs that can be scheduled for user i

O matrix of network output volume

P_α set of unit computational resource prices

P_β set of unit data storage prices

P_θ set of unit output bandwidth prices

P_ρ set of unit input bandwidth prices

P_c total computing price

P_d set of new prices generated by our pricing methodology

P_e set of prices the cloud provider charges

P_i total inbound bandwidth price

P_n over all network price

P_o total outbound bandwidth price

P_s total storage price

Q_i total message inbound bandwidth price

Q_o total message outbound bandwidth price

R_d set of revenue with the prices generated by our pricing methodology

R_e set of revenue a cloud provider receives with existing unit prices

R_i total inbound bandwidth price

R_o total outbound bandwidth price

S set of servers the cloud system contains

S_r vector of resource types each server contains

T_s vector of number of Virtual Machines submitted to the system by all tenants

T_n vector of number of Virtual Machines allocated to a user

U set of cloud tenants active in the cloud system

V_r matrix of network volume data reading from one cloud service

V_m matrix of network volume with inter-cloud message exchange

V_s matrix of network volume of state-transition

Acknowledgments

I would like to take this chance to thank all of those who have helped me and supported me during the course of my research work.

I want to appreciate Professor Victor C.M. Leung for providing me the life changing chance of pursuing a Graduate Degree and conducting research in Cloud Computing. Without your mentorship, guidance and support, this work would not be possible. Also, I am thankful to Professor Abdalalh Shami for the help of the manuscript of the published paper contained in this thesis.

I would like to thank my colleagues Xiuhua Li, Xiaofei Wang and Wei Cai for the help during my research journey. Thank you all for the effort of discussing research topics and tips with me.

This work was supported by the Natural Sciences and Engineering Research Council (NSERC) of Canada.

Dedication

Chapter 1

Introduction

This thesis presents how resource management of the cloud system can be accomplished through demand control using dynamic prices. As more and more companies or individuals have moved their applications or computing tasks to the cloud, cloud providers are facing a new challenge: the server utilization is only at 10% [1]. Low server utilization has numerous causes, such as uneven application fit where the application cannot fully utilize the resources allocated or the uncertainty in demand forecasts that is introduced by the dramatically varied demand of the cloud resource between peak and non-peak periods. While many research works are devoted to optimize the resource allocation techniques in the effort of achieving higher server utilization, how to control resource demand so the correct level of resource provisioning can be determined has become the next research question. This thesis introduces a new approach of using dynamic pricing to control aggregate demand to avoid resource over/under-provisioning and achieve a higher profitability. Further, to avoid requiring users to pay close attentions to the frequent changing prices and make real-time request decisions themselves, this thesis demonstrates a price-aware cloud federation system that could cognitively reacts to the dynamic prices and automatically makes demand decisions.

1.1 Motivation

For any for-profit organization, the ultimate goal is to maximize its profitability. Ever since the Cloud Computing has emerged, we have observed a rising number of cloud service providers providing different service types, such as infrastructure, computing and storage, have been introduced to the market [2]. To achieve its ultimate goal, a cloud provider is devoted to seek the way to lower costs through resource management and develop an efficient pricing strategy that focuses not only on how much it increases price, but also on how much it increases profitability[3].The virtualization technique is the key in cloud computing. Computational and storage resources are provided by Infrastructure as a Service (IaaS) cloud through different types of *Instance*s. Upon the request of an *Instance* by a tenant, if the cloud has enough resources to host the instance, a virtual machine (VM) is allocated onto a server so that the cloud tenant could run her applications or other computational tasks on the *Instance*, or the VM to be specific. Therefore, resource management in cloud computing is well studied in the perspectives of virtualization and allocation optimization [4][5][6]. However, optimization from any aspect alone is limiting. The amount of resources a cloud tenant needs varies from time to time. Traditional resource allocation and provisioning techniques still require data centers to be prepared for the intense resource demand during peak period [7]. Incorrect estimations of user demand levels may lead to costly over-provisioning of resources. Moreover, regardless of how the cloud is considered to be an unlimited resource pool, any resource has fixed capacity. In case of resource competition, the cloud resource should be shared among all users fairly. It is important to incentivize cloud tenants to request for cloud resources reasonably. Also, given the cloud system is becoming more compatible and easier to access, changing to a cheaper service provider has becoming a common practice for users who are price sensitive. Thus, the demand of cloud resources is more reactive nowadays. A competing cloud provider is urged to develop a more "dynamic" pricing strategy that could control the reactive demand to maintain system utilization level and revenue. Therefore, user demand patterns should also be considered as inputs to the VM placement problem. In this thesis, we propose a pricing methodology that, by shaping the demand pattern, contributes in the resource management perspective that achieves system resource high utilization with Knapsack Algorithm and resource fair-sharing with Heterogeneous Dominate Resource Fairness (DRFH) Algorithm. The profitability of the proposed pricing methodology is evaluated through numerical and trace-driven simulations. Results have shown that it enhances the cloud provider's revenue. The ability of the proposed pricing methodology to induce user demand pattern is also proven to be a success.

On the other hand, the key for a service to be acceptable for general users is powerful functions yet simplified usages. The pricing methodology we proposed may need users constantly paying attentions to the current prices a cloud provider charges for their VMs. Users also need to constantly make decisions about whether or not to continue using the service at the given prices since the price may be changing. These may become new obstacles for users to adopt the cloud services. Moreover, the goal of service consumers, especially consumers with huge service demand, are to receive qualified services with minimum costs possible. To this end, we introduce a new way to improve user experiences on such cloud services. We propose a price-aware cloud federation system that provides the capability of automatically choosing and migrating user tasks to the cloud system that is charging at the lowest rate on real-time basis. The cloud federation [8] system inter-connects global "cloud of clouds" with the ability to communicate and cooperate between clouds provided by different vendors. Our pricing methodology is more user-friendly with the cloud federation system. When one cloud resource is too expensive for a user, the cloud federation system would automatically migrates her program or data set running on a VM of a cloud service to a cloud service with an affordable rate. Simulation results have shown that the total cost for users is dropped significantly with the cloud federation system, comparing to a single-cloud service.

The rest of the thesis is organized as follows: in Chapter 2, we provide an overview of different techniques of resource management, pricing-based demand control and cloud federations in cloud computing. In Chapter 3, we propose a pricing methodology that enhances the cloud providers' revenues and keeps system resource utilization high. We study the ability of the proposed pricing methodology to affect the user request submission pattern and the profitability of the proposed pricing methodology for a cloud provider through numerical and trace-driven simulations. In Chapter 4, we introduce a price-aware cognitive cloud federation system as a supporting tool of our proposed pricing methodology. In Chapter 5, we discuss the conclusion and suggestions of possible extension of the work included in this thesis.

1.2 Summary of Results and Contribution

In this thesis, we

- Propose a pricing methodology that leverages cloud task scheduling algorithms to determine the optimal demand pattern of the cloud service, generates dynamic prices according to current workload of the cloud system and uses pricing as a tool to control the aggregate demand and achieve a higher profitability.

- Propose a framework of a price-aware cloud federation system as a supporting tool for the proposed pricing methodology that would help user automatically choosing, and migrate computation tasks to, a cloud service that is charging at a affordable rate.
- Provide a model of price-aware cognitive decision making and mathematically formulation of the optimization problem.

Chapter 2

Background

2.1 Resource Management

A cost efficient cloud need to use fixed costs optimally and to discourage behavior that drives excessive service costs. Unlike traditional enterprises where leading cost is operational staff, the leading cost of a cloud data center is servers and infrastructures, facilities dedicated to consistent power delivery and evacuate heat [1]. Study shows that 45% of the amortized cost goes to servers and 25% of the amortized cost goes to infrastructure. So one of the goals of the cloud provider is to seek a high rate of return of its investment. Unfortunately, statistics show that server utilization is only at 10% [1]. Low server utilization has many causes. Processing capacity is over-provisioned for many business applications to meet the Service Level Agreement (SLA) specifications in terms of appropriate level of quality, availability, reliability and performance during peak demand periods. Also, to guarantee application isolation, the traditional ad-hoc deployment of one application requires one application per unit of physical hardware [4]. Lots of server resources such as central processing unit (CPU), Memory and storage unit are wasted during low demand periods. Furthermore, the networking costs concentrate mainly in the networking gear (i.e. switches, routers and load balancers), network power consumption and network traffics between geographically distributed data centers and the Internet Service Providers. Therefore, it is obvious that both achieving high server utilization and finding the optimal placement of data centers to deploy service applications have a role to play in achieving cost efficient cloud. Our dynamic pricing methodology utilizes a resource allocation algorithm to determine how many VMs each user should receive according to their requests. In this section, we discuss resource allocation optimizations which have been tackled from various perspectives.

Static and Dynamic Virtual Machine Placement

In data centers that use static VM placement algorithms, a physical machine (PM) is chosen at the time of VM creation with the knowledge of PM capacity and the resource requirements of the VM. The placement might not be rearranged until the PM is restarted [9]. On the other hand, dynamic placement algorithms leverages the VM live migration capability to rearrange the VM placement in response of variations in traffic load, resource demand and unplanned downtime of a PM. The fact that VM placement decisions are made with the knowledge of system status such as PM capacity and traffic load enabled our pricing methodology to calculate new unit prices according to real-time system load.

SLA Violation Reduction Virtual Machine Placement

SLA violation reduction VM placement algorithm reduce SLA violations. SLA specifies not only level of qualities of the cloud services that the cloud provider provisions, but also has additional constraints regarding security, data availability or the physical location of the data storage. SLA is negotiated between cloud providers and consumers [10]. A consumer may require that the VM cannot run on a PM that is located over-seas. A cloud provider may require that the resource demand does not overload the resource capacity by a probability of p. [4] proposed a dynamic VM placement algorithm that minimizes the number of PMs required to provision the VMs subject to constraint of the probability of server overloading, which reduces the rate of SLA violations. It measures the time series and probability distribution of VM's CPU demand during the interval t. The CPU demand for the next interval of length t is predicted based on the historical demand during the prior interval of length t. VM placement or VM migration decisions are then made based on the prediction. This algorithm fulfills the resource demand with a probability.

Another VM placement algorithm minimizes the number of resource utilized, SLA violations due to the failure of allocating all VMs and the allocation time [11]. All PMs are organized in a binary search tree. Searching for a PM involves a binary search that would reduce the VM placement computation time. A VM scheduler calculates the ratio of requested VM specification to the available PM specification for each PM in the binary tree until it finds the best fit (PM that has the maximum VM specification to PM specification ratio smaller than 1). VM migration also requires the destination PM to have enough computation resources that could host the migrating VM. Therefore, migration domain is carefully defined and the placement service can associate different policies with different migration domains.

A Constraint Programming optimization approach with a global utility function integrates both the degree of SLA fulfillment and the operating costs [12]. The VM placement stage is separated from the VM provisioning stage within the global decision layer autonomic loop. Both problems are formulated as Constraint Satisfaction Problems (CSP), as well as are instances of an non-deterministic polynomial-time hard (NP-hard) knapsack problem. Such an approach avoids the problems encountered by rule- and policy- based systems where conflicting objectives must be handled in an ad-hoc manner by the administrator.

Network-aware Virtual Machine Placement

Optimization of the network costs is another resource management optimization goal. In [5], a placement algorithm focused on minimizing communication cost among VMs is proposed. With one CPU/Memory unit on a host is referred to as a slot, VM can be placed in any slots available. The algorithm first partitions VMs into clusters using min-cut graph algorithm to ensure communicating VM pairs are partitioned into the same VM cluster. Available slots are partitioned with slot pairs with low-cost connections in the same slot cluster based on the cost matrix. Then, the algorithm uses C_{ij} to denote the communication cost between slots i,j. D_{ij} denotes the traffic rate from VM i to j. e_i and g_i denotes the external traffic for VM i and communication cost between VM i to the gateway. With the assumption that there are n VMs and n slots, the goal of the algorithm is to find the permutation function that maps VM clusters with heavy communication traffic to slot clusters with low-cost connections.

Network-aware VM placement can also focuses on network power reduction with consideration of account network topology and network traffic [13]. The VM placement problem is formulated as a network flow problem using a weighted directed graph $G = (V, E)$. Switches, PMs and client applications are modeled as a vertex v, an edge e represents communication links between PMs and switches. The source or destination of the flow network is one of VMs or client applications. An edge has a non-zero flow only if the switches or PMs at both ends of the edge are powered on. The flow assignment on each edge specifies the amount of traffic flow for every source-destination demand. Then the optimal VM placement is found to be the shortest path in the flow network. The algorithm keeps iterate to select a VM placement requires minimum increase in network power as a locally optimal placement.

With considerations of both constraints on local physical resources, such as CPU and Memory, and constraints on network resources evolving from complex network topologies and dynamic routing schemas, network-aware VM placement optimality problem is much more complex due to its quadratic nature and since it involves many factors beyond the physical host [14]. This problem strives to minimize the maximum ratio of the demand and the capacity across all cuts in the network, in order to find placement solutions that, by having spare capacity on each network cut, can absorb unpredicted traffic bursts. Since this problem is an NP-hard problem, the solving algorithm uses two novel heuristics, with diverse tradeoffs between solution quality and execution time. The first heuristic exploits integer programming techniques to solve the problem. It exploits the tree network structure to define and solve small problem instances on one-level trees recursively. Thus, each placement step, solved by a mixed integer programming solver, has to deal with a reduced number of VMs, hosts, and network topology cuts. The second heuristic, which completely leaves out mathematical programming, consists of two main phases: the first one ranks all the traffic demands, while the second one exploits the ranking to place VMs on available hosts.

2.2 Fairness in Resource Management

Above mentioned algorithms optimize resource allocation solely from the cloud providers' perspective. Since the cloud is a resource pool with fixed capacity, competition of resources among users will arise in a demand burst. In this section, we discuss resource allocation techniques taking fairness among all users as an optimization goal and the fairness-efficiency trade-offs.

Hadoop Fair Scheduler and Quincy are slot-based fair scheduling falls in the category of Asset Fairness, where each user receives equal share of resources [1]. Unlike original Hadoop scheduler that put

[1] http://hadoop.apache.org/docs/r2.7.0/hadoop-yarn/hadoop-yarn-site/FairScheduler.html

applications in a queue when the resource is fully occupied, the fair scheduler frees up resources for the newly joined applications, so that each application in the system eventually occupies the same amount of resource.

Considering the multi-resource nature of cloud platform, [15] introduced the idea of Dominant Resource Fairness (DRF) to this area. DRF is a multi-resource allocation problem with the goal to equalize the dominant share, which is the maximum ratio of any resource allocated to a user in the system, among all cloud tenants. It is modeled as a max-min optimization problem and is shown to possess a number of fairness properties such as sharing incentive, strategy-proofness, envy-freeness and Pareto efficiency. A higher system utilization is also promised compared to a slot-based fair scheduling and a max-min fair sharing algorithm that focuses on a single resource type [15]. Yet one limitation of DRF is that it simplifies resources in a cloud computing system as resources in a super computer. It does not consider the capacity limitation of each server and the fact that, after several VMs are placed in a server, the remaining resources of this server might not be sufficient to host other VMs. DRFH is proposed in [16], which generalizes DRF [15] by applying it to real cloud computing systems with heterogeneous servers.

For resource competitions during a congested situation, [17] has indicated that it is unfair for a specific user to be given a small amount of resources, or experiences a long service completion time. In the method proposed in this work, when submitting a task request, users are required to submit a reduction ratio specifying how much the requested size of resource can be reduced in the case of congestion. Then, the system dynamically adjusts the resource allocation according to reduction ratios for each user when the resource utilization level is greater or equal to a predefined threshold. Simulation results show that the proposed method achieves the fair allocation.

Fairness-efficiency trade-offs were studied in [18] with multiple resource types. The authors defined percentage efficiency as the percentage difference between the total number of jobs processed in a given allocation and the maximum number of jobs that can be processed with the same capacity constraint, and the leftover capacity as the amount of unused resources. They proposed and evaluated two fairness functions for multi-resource allocations. Their evaluation results show that fairness functions bear different fairness-efficiency trade-offs.

2.3 Price-based Demand Control

Despite all these effort spent on resource allocation of the cloud system, the resource over/under provisioning problem still cannot be solved since the demand is still unpredictable. The right pricing heavily affects customer behavior, customer commitment, and thus the organizations' success [19]. Use pricing to guide user behavior toward a more efficient operating point has been shown to be a success in radio resource management, wireless networks [20] and the Internet [21]. In this section, we discuss price-based demand control.

Demand Control with Dynamic-Rate Pricing

The smart metering model proposed in [22] provides dynamic pricing of the cloud service based on the load condition. The resource usage is metered and recorded. The customer's historical utilization statistics are used to predict the load condition for the next time interval of operation and thus determine the price. The price is then published on the cloud provider's website so the customer could decide whether to continue her usage.

Pricing plan can be used to incentivize users to shift their usage from peak to off-peak periods by providing lower prices in less-congested periods [21]. Authors have implemented a mobile application and invited 50 trial participants to evaluate their proposed model and user sensitivity to prices. They conclude that users did shift their traffic from high- to low- price periods.

Auction game-based pricing model has also been studied for the cloud market. A double auction bayesian game-based pricing model proposed in [23] provides the capacity for users to trade their left-over cloud resources. Both cloud resource providers and consumers form a double auction. Resource allocation takes place when two sides make a deal at a certain price.

Demand Control with Fairness-Aware Pricing

While the resources of an instance are dedicated to one cloud tenant, the cloud bandwidth is shared among all cloud tenants. Haiying et al. [24] introduced a pricing policy that focuses on preventing users from competing for bandwidth at the cost of the data center or other users with current flat-rate pricing models. Their pricing policy achieves network proportionality, which means that network resources allocated to tenants are proportional to their payments and thus achieves fairness between tenants. Another work [25] considered the price competitions among different providers in the open market. They tackled the cloud competition problem and proposed a non-cooperative game to investigate the price competition among cloud providers and its impact on profit of all cloud providers, tenant satisfaction and final instance prices.

2.4 Pricing Models of The Cloud Service

In this section, we discuss existing pricing models of the cloud service. For IaaS cloud, computing infrastructures such as CPUs, storage and network are virtualized and provisioned to cloud tenants as *Instance*s. Each cloud tenant simply requests *Instance*s to run her applications or other services. Some examples of IaaS are Amazon Elastic Compute Cloud (Amazon EC2) and Google Compute Engine [19]. The flat rate pricing model is used by most of the cloud providers, such that cloud tenants are charged according to the time of usage regardless of network congestions or system workload. Many researchers have proposed pricing models that provide dynamic rates to encourage cloud tenants to lease *Instance*s in a desired manner.

Computational Pricing

Since our pricing methodology optimizes computational resources, in this section we discuss existing computational pricing of the cloud services. Reserved Instances: Amazon EC2 provides the option for customers to subscribe to its cloud service for one- or three-year periods with a non-refundable one-time payment at a lower rate. Cloud tenants can use their reserved resources at anytime during the reserved period while Amazon ensures that the reserved resources are always available [26]. However, resources may be wasted if the cloud tenant has reserved an instance, but for most of the time left it idle. Freemium and Usage-based: To encourage potential cloud tenants to try their cloud services, both Amazon EC2 and Google Compute Engine provide free but limited amount of resources for a limited time period. Once the actual usage exceeds the limits, usage-based pricing is generally applied. Usage-based pricing is the most common pricing model for cloud services as it is elastic and charges a tenant based on the actual usage. A typical Google Compute Engine standard instance may contain 1 virtual core, 3.75GB memory and charges in minute-level increments for the time that the cloud tenant runs her instance[1]. Amazon EC2 bills to the nearest server-hour or gigabyte-month [26]. Usage-based pricing allows users to use the cloud service anytime without a long-term commitment, but the access to cloud service is not always guaranteed. Financial Option: Sharma et al. [27] proposed a pricing model that employs the financial option theory and Moore's law. They treated the cloud computing commodities as assets and mapped cloud parameters to the Black-Scholes-Merton Model. Moore's law is used to describe the value of the resources in cloud. The price determined by their model is the optimal price the cloud provider should charge the clients to recover its initial cost.

Although flat-rate pricing is widely adopted by mobile, Internet and television service providers, [28] has shown that flat-rate pricing is optimal when the capacity or quantity of the expected sales is infinite. On the other hand, the dynamic pricing method has shown its benefit when the capacity is fixed and unsold volume is worthless such as in the airline, hotel and electric utility industries [29]. Spot Instances: Amazon provides its unsold cloud capacity as spot instances for customers to bid on. Amazon sets its spot prices through a market-driven auction and publishes its spot price for the next time period online so that customers can run those instances as long as their bids exceed the current spot price. Spot Instance pricing allows Amazon to sell more of its unused resources at the highest possible rate while preserving control over the spot price [30]. Unfortunately, for cloud tenants, spot instances introduce uncertainty to their access to the cloud service because they may be terminated at any time.

Networking Pricing

Despite various types of pricing policies, almost all networking pricing models follow a principle that charge users' Internet usage bidirectional with different standards[2]. In general, ingress is cheaper than egress. Despite various types of pricing models, almost no pricing model comes with a performance guarantee. Google advertises that its Google Compute Engine instances can achieve maximum perfor-

[2] https://cloud.google.com/products/compute-engine/

mance, throughput and availability at low cost through its native load-balancing technology [3]. However, what kind of maximum performance and how low the cost are not specified anywhere. The Amazon EC2 SLA specifies its service commitment of 99.95% availability [4]. Yet no performance guarantee is included in the SLA.

Storage Pricing

Storage pricing provided by various cloud services are relatively lower compared to computing and networking. Microsoft Azure [5] provides four types of storage, including block blobs, page blobs and disks, tables and queues and files (in preview stage). It also has four redundancy levels, including locally redundant, zone redundant, geographically redundant and read-access geographically redundant. In addition to basic storage price, some cloud companies also charge users for data operations (request, delete, etc.) and data transfer, e.g., Amazon Simple Storage Service and Google Cloud Storage. In this work, we simply use network transmission to represent these additional cost applied to the storages.

2.5 Service Recommendation Systems

Because each cloud service provides different VM types and SLAs, selecting an appropriate cloud service for user applications may require some technical background. To ease up this process, some research works have been devoted to service recommendation systems [31][32][33]. Authors in [31] proposed a business-centric revenue-driven recommendation system that considers the appropriate recommendation time, price and the number of repetitions when finding the optimal recommendation strategy to maximize the seller's expected total revenue. In [32], authors proposed a recommendation system for IaaS Cloud applications based on analytic hierarchy process method. The proposed system automatically selects a cloud service that satisfies the required network Quality of Service (QoS) constraints on real-time basis. In [33] authors proposed a cloud service discovery system that automatically find a cloud service according to user provided service query specified in pre-defined query structures.

[3] https://cloud.google.com/products/compute-engine/

[4] http://aws.amazon.com/ec2/sla/

[5] https://azure.microsoft.com

2.6 Cloud Federation Systems

The term of cloud federation, or inter-cloud, the cloud of clouds, was first introduced by Kevin Kelly in 2007. The inter-cloud is analogous in the way the Internet works. An Internet service provider that has an endpoint attached to it will access or deliver traffic from/to source/destination addresses outside of its service area by using Internet routing protocols with other Internet service providers with pre-arranged exchange or peering relationships. The work [34] first proposed the inter-cloud blueprint to describe the high level architecture of the inter-operating of multiple clouds. With the concept of inter-cloud, distributed software systems [35] explored a new application scenario. Federated cloud system helps to achieve better QoS, reliability and flexibility [36]. Authors in [37] proposed a cloud federation system that provides profit-aware solutions in order to receive benefits from pricing policies provided by multiple clouds. Some of the other cloud federation systems are proposed for load balancing and distribution of elastic applications among different cloud data centers to achieve reasonable QoS levels [38].

Program Decomposition

Decomposition of software program is breaking the system into small parts that are easy to maintain. The decomposition can be in terms of system functions as classes or objects and data entities. Program decomposition, distributed storage and computation are the fundamental techniques for parallel computing, which is widely adopted as a solution in big data processing and complex computing [39]. For example, Hadoop decomposes data into chunks and distributes them among computer clusters. Then, based on the data each node contains, the Hadoop MapReduce sends appropriate program module to each node to process data in parallel [6]. [40] proposed a decomposition algorithm that decompose a sequential program into speculatively parallel threads that can run on multi-processor chip, with considerations of data dependency and load imbalance. In [41], authors proposed a program decomposition framework that provides near-optimal mappings of program segments to machines with minimum-cost. Data elements needed for the program segments can be transferred among machines. The cost of exe-

[6] https://en.wikipedia.org/wiki/Apache_Hadoop

cuting a program segment depends on machine selections and associated data transfer costs.

With decomposed and distributed programs, cognitive optimization of resource allocation is also an open issue. Intrinsically as a group of dynamic partitioning problem, research on the dynamic partitioning between cloud and users' mobile terminal has been conducted from the perspectives of offline K-step approach [42] and flexible partitioning [43]. A similar idea has been used in [44], which has designed and developed a cognitive platform that enables task migration and dynamic task allocation between the cloud server and the devices.

Privacy Regulation Through Across-Clouds Distribution

Keeping computing tasks that involve sensitive data in a private cloud and outsourcing the rest of computing tasks that involve insensitive data to public clouds is a preliminary solution for data privacy regulation. However, as more data-intensive computing tasks are required, this kind of hybrid cloud computing has became inappropriate. [45] proposed a Privacy-Aware Data Intensive Computing on Hybrid Clouds that automatically splits data-intensive tasks according to security levels of the data. They modified MapReduce's distributed file system to replicate data and send sanitized data to the public cloud. Using multiple public clouds to ensure data security is also a popular solution. Faults in software or hardware in cloud computing are known as Byzantine faults. Many research work has been done on Byzantine fault tolerance [46]. DepSky virtual storage cloud system [47] leverages the Byzantine quorum system protocols to ensure data security. The DepSky system is consisted of *n* clouds, while the DepSky system reads and write to each cloud separately. With intensive tests, experiences of working on a specific cloud and extraordinary positive user reviews, a cloud provider might be trustworthy. Besides, a series of redundant array of independent disks (RAID) of the cloud work [48] tackle the problem by encrypting and encoding the original data and later by distributing the fragments transparently across multiple providers. This way, none of the storage vendors can see the full picture of the client's data.

Chapter 3

A Pricing Methodology for Revenue Enhancement

3.1 The Pricing Methodology

As discussed in Chapter 1, in this chapter, we propose a pricing methodology with dynamic pricing [49], which has been shown to be beneficial in many industries [29]. Under this methodology, an IaaS cloud provider may charge different tenants different prices. Inspired by the principle of the *Law of Supply and Demand*, we aim to use pricing to incentivize cloud tenants to use data center resources in a way that high utilization level is ensured. Given that the types of VMs and the number of VMs requested by all active cloud tenants are known at any point in time, our pricing methodology leverages a task scheduling algorithm to evaluate how many VMs each tenant should receive so that the system utilization is maximized. Since the fewer resources allocated to a user by the task scheduling algorithm indicates that the system has more difficulty to place the user requested VM, we are motivated to raise the prices of these users who overload the system. Thus, a pricing weight for each tenant is derived according to the number of VMs allocated to the tenant. Finally, new prices are determined based on the prices the cloud provider was charging originally and the weight derived in two methods: we study our pricing methodology with Total Unit Price Redistribution (TUPR), where we sum up the unit prices charged by a cloud provider originally and redistribute the charges among users according to the pricing weight, and Total Revenue Redistribution (TRR), where we sum up the revenue received by a cloud provider originally and redistribute the charges among users according to the pricing weight.

Moreover, we study our pricing methodology with two resource allocation algorithms. First, we study our pricing methodology with Knapsack Algorithm, a combinatorial optimization algorithm that aims to maximize the total revenue and the server utilization level of a cloud data center. From the simulation results we notice that some of the users may not be getting any resources and this is not fair. Therefore, we further study our pricing methodology with the DRF algorithm [15] where total resources allocated to each cloud tenant is fair and resource utilization is higher compared to other fair sharing resource allocation algorithms [50]. As a result of price changes, requests for a specific type of VM by existing or potential users are encouraged or discouraged according to the *Law of Supply and Demand*, so system optimal resource utilization and optimal usage behaviors are eventually achieved. From simulation results, we show that the total revenue a cloud provider receives is enhanced and overall fairness is archived among users through simulations. The proposed pricing methodology follows a pay-as-you-go pricing model. It suits the fundamental characteristics of the cloud as an on-demand and usage-based service. The proposed pricing methodology is transparent and truthful because it takes the total allocable resources into consideration so that any manipulation from the backend such as resource throttling or capacity right-sizing is reflected in the new prices. Unlike the Spot Instance [30], our proposed pricing methodology does not produce service termination. Also, instead of setting the price to the highest possible rate, the prices generated by our pricing methodology are determined by looking at the utilization of system resources.

3.1.1 Methodology

As mentioned above, the main motivation of this work is leveraging pricing policy to manipulate user behavior, for the purpose of increasing system resource utilization. Figure 3.1 is a sequence diagram that demonstrates the working mechanism of the proposed Fairness-aware Pricing Methodology. Users request VMs with known current prices of VM types. Then, upon the arrival of a new user's VM request, task scheduling algorithms are executed to dynamically allocate cloud resources, subject to the cloud server's existing tasks and available resources. The algorithm not only determines how easily the VM could be hosted, but also decides on which server the VM is to be placed. While the user's request is being fulfilled, the real-time cloud information, including incoming VM settings and current cloud workload, are updated to the decision making module that implements our proposed pricing model. The new price is generated accordingly after the allocation decision. With the new prices, existing users react by increasing or decreasing their usage percentage of the cloud service. Once the user reactions are submitted to our system, the algorithm starts again from the beginning as a new loop. Also, potential users may decide to start to use the cloud service. This procedure continues as the users' demands fluctuate. It is apparent that a decision making module that predicts users' reaction to future prices and subsequently adjusts prices to achieve predefined optimization goals is the key of our methodology. Details are described in following sections.

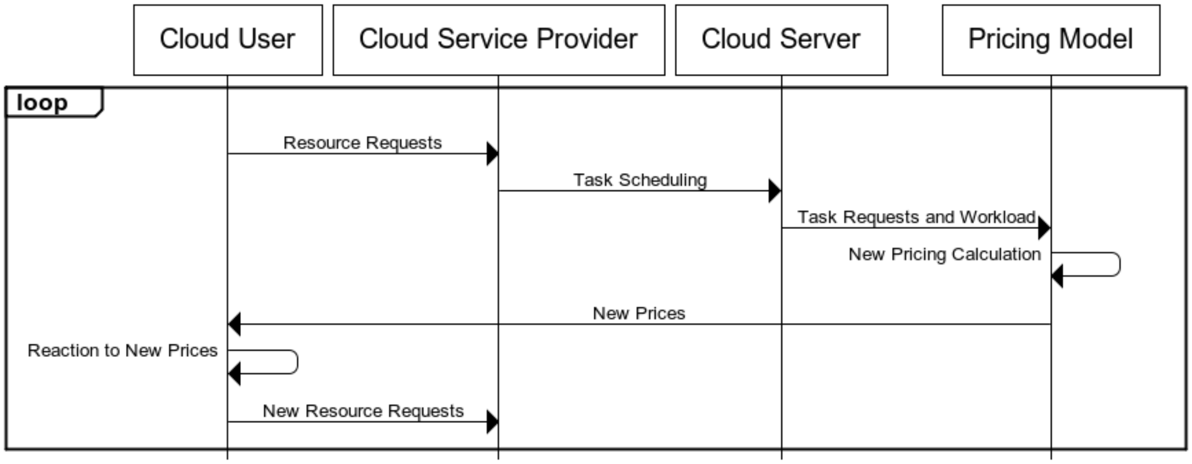

Figure 3.1: Sequence Diagram for Fairness-aware Pricing

3.2 Problem Formulation

With the pricing methodology proposed in the last section, a number of open issues remain to be addressed before the methodology can be put in practice. For instance, on which server or servers should the cloud system place the requested VMs? How would pricing affect users' demands? How to determine the price, in order to enhance the revenue? In this section, we discuss these issues.

We define the input variables as shown in Table 3.1. Suppose the set of servers S, resource types S_r and resource capacity vector c_l are known for each server l. Also suppose the set of cloud tenants U, resource vector of their desired types of VM D_i and the number of VMs users initially requested T_s are known.

$S = \{1, \ldots, k\}$	Set of servers the cloud system contains.
$S_r = \{1, \ldots, m\}$	Resource types each server contains.
$c_l = \{c_{l1}, \ldots, c_{lm}\}$	Normalized resource capacity vector for each server l, where c_{lr} denotes the total amount of resource r available in server l.
$U = \{1, \ldots, n\}$	Set of cloud tenants active in the cloud system.
$D_i = \{D_{i1}, \ldots, D_{im}\}$	Resource vector of the VM requested by user i, where D_{ir} denotes the fraction of resource r required by user i's VM.
$T_s = \{T_{s_1}, \ldots, T_{s_n}\}$	The number of VMs submitted to the system by all tenants, where T_{s_i} denotes the number of VMs submitted by user i.

Table 3.1: Definition of Input Variables

3.2.1 Price Sensitivity

Our pricing methodology leverages the concept of price sensitivity to model user reaction to the new price generated by our pricing model. In a competitive market, price sensitivity defines the highest price a customer would pay for the desired product and the lowest price a customer would pay without second thought of the product quality [51][52]. As the most powerful tool to marketers, price sensitivity is well studied when setting the price of a new product to maximize the demand of the product and the business outcome [53]. For a majority of the buyers, the price is not only a key factor that will influence their purchase decisions, but also an indicator for them to perceive product or service quality. The price threshold that captures consumer insensitivity to small price changes was examined in [53]. Harmon et al. [51] studied the Price Sensitivity Measurement (PSM) model and incorporated it into the Value-Based Software Engineering (VBSE) process. PSM check is used twice during the VBSE process to first refine customer value assessment of the potential product and then help in finalizing the development of a marketing plan prior to commercialization. Moreover, the Law of Supply and Demand suggests that the availability and desirability of a product has a great effect on the product price [54]. If the supply is sufficient but the demand is low, the price will be low. In contrast, if the supply is inadequate but the demand is high, the price will be high. Then, given that all other factors are equal, the demand of the good or service decreases as the price increases[7]. The demand curve of a commodity is downward sloped as shown in Figure 3.2. However, to the best of our knowledge, the specific demand curves of cloud services have not been well studied.

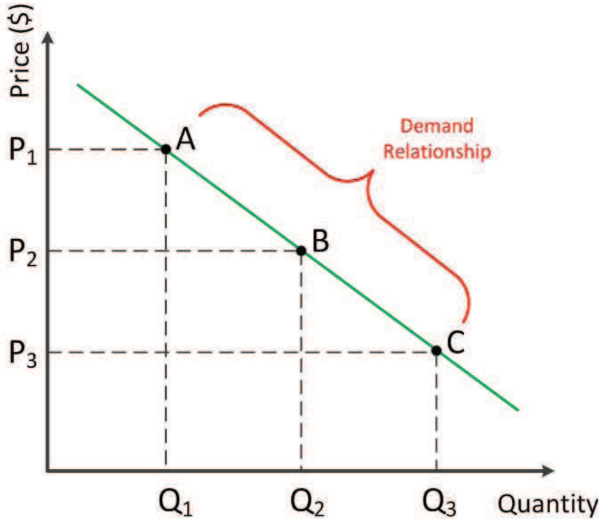

Figure 3.2: Commodity Demand Curve

Since the supply and demand functions of the cloud service as the price changes has not been well studied, in this work, we utilize an iso-elastic demand function to model how the level of user demand changes as price changes [55]

$$\begin{aligned} f_R(p) &= (1/p)^{(1/\alpha)}, \alpha \in (0,1) \\ \alpha &= \frac{1}{\log_{1/p} f_R(p)} \end{aligned} \tag{3.1}$$

where α is the elasticity coefficient and $f_R(p)$ represents the demand.

In this work, a higher demand level indicates an increasing need of cloud VMs.

[7] http://www.investopedia.com/terms/l/lawofdemand.asp

3.2.2 Cloud Task Scheduling

One of the most important steps of our proposed pricing methodology is to determine how easily a VM can be handled by the cloud. The computing resources provided by the cloud consist of heterogeneous servers where each server may contain a different amount of resources. Hence, the allocation of various forthcoming VM requests onto the distributed servers in the cloud is of great importance for the overall system efficiency. In this work, we utilize resource allocation algorithms that maximizes the cloud server utilization level to help us determine how should the user requests of VM should be addressed. Namely, given the number of VMs each user has initially requested, T_{s_i}, we determine T_{n_i}, the number of VMs each user is allocated in the cloud system by a cloud task scheduling algorithm.

3.2.3 Revenue Enhancement

One of the goals of our proposed pricing methodology is to enhance the revenue for cloud service providers. According to the *Law of Supply and Demand*, given that other factors such as advertisement, brand preferences, product differentiation and segment membership are kept the same, the quantity demanded and the price of a commodity are inversely related [51]. For simplicity, let

$$w_i = \frac{\frac{1}{T_{n_i}}}{\sum_{i \in U} \frac{1}{T_{n_i}}}, \forall i \in U \tag{3.2}$$

where $w = \{w_1, \dots, w_n\}$ denote the weights assigned to n tenants.

Revenue Enhancement with Total Unit Price Redistribution

To calculate our new prices of a VM, we first study the case in which we keep the total unit price of all users the same and redistribute the sum among users according to the weights. To be specific, the new price for each user i is calculated according to the weight of user i and the price the cloud provider charges each tenant. For example, a typical Google Compute Engine standard instance may contain 1 virtual core, 3.75GB memory and charges in minute-level increments for the time the cloud tenant runs her instance at \$0.07/hour[1]. Let $P_e = \{P_{e_1}, \dots, P_{e_n}\}$ denote the prices the cloud provider charges n tenants and $P_d = \{P_{d_1}, \dots, P_{d_n}\}$ denote the new prices generated by our pricing model for the n tenants, then,

$$P_{d_i} = w_i \times \sum_{i \in U} P_{e_i}, \forall i \in U. \tag{3.3}$$

Algorithm 1 Pricing Calculator for user *i* with Total Unit Price Redistribution

```
Set $P_{e_{sum}} \leftarrow 0$
Get $T_{n_i} \leftarrow TaskSchedulingAlgorithm$
for $n = 1$ to $N$ do
    $w_i \leftarrow \frac{\frac{1}{T_{n_i}}}{\sum_{i \in U} \frac{1}{T_{n_i}}}$
    $P_{e_{sum}} \leftarrow P_{e_{sum}} + P_{e_i}$
end for
$P_{d_i} \leftarrow w_i \times P_{e_{sum}}$
```

The algorithm for calculating unit prices for user *i* is shown in Algorithm 1.

Revenue Enhancement with Total Revenue Redistribution

Next, we study the case in which we keep the total revenue of all users the same and redistribute the sum among users according to the weights, given the number of VMs that can be scheduled for each user by DRFH. To be specific, the revenue generated by user i, R_{e_i}, is calculated as

$$R_{e_i} = T_{n_i} \times P_{e_i}, \forall i \in U. \tag{3.4}$$

New unit price $P_d = \{P_{d_1}, \ldots, P_{d_n}\}$ is calculated as

$$P_{d_i} = \frac{w_i \times \sum_{i \in U} R_{e_i}}{T_{n_i}}, \forall i \in U. \tag{3.5}$$

Algorithm 2 Pricing Calculator for user *i* with Total Revenue Redistribution

```
Set $P_{e_{sum}} \leftarrow 0$
Get $T_{n_i} \leftarrow TaskSchedulingAlgorithm$
for $n = 1$ to $N$ do
    $w_i \leftarrow \frac{\frac{1}{N_i(A_i)}}{\sum_{i \in U} \frac{1}{N_i(A_i)}}$
    $R_{e_{sum}} \leftarrow R_{e_{sum}} + P_{e_i} \times N_i(A_i)$
end for
$P_{d_i} \leftarrow \frac{w_i \times P_{e_{sum}}}{N_i(A_i)}$
```

The algorithm for calculating unit prices for user *i* is shown in Algorithm 2.

Reactive Revenue Calculation

Then, given P_{e_i} known, the total revenue R_e a cloud provider receives is calculated as

$$R_e = \sum_{i \in U} T_{s_i} \times P_{e_i}, \forall i \in U. \tag{3.6}$$

User elasticity is determined through the demand function

$$T_{s_i} = f_R(P_{e_i}), \forall i \in U. \tag{3.7}$$

We input the same user demand vectors and system resource vectors into our pricing model to obtain the new prices P_{d_i}. Then, we use the demand function again to derive the number of VMs users would request after they receive the new prices, denotes as T'_{s_i}

$$T'_{s_i} = f_R(P_{d_i}), \forall i \in U. \tag{3.8}$$

All users' VM requests T'_{s_i} are scheduled using first-in-first-out (FIFO) scheduling again. Let T'_{n_i} be the number of VMs each user is allocated. The total revenue R_d a cloud provider receives with the new prices is calculated as

$$R_d = \sum_{i \in U} T'_{n_i} \times P_{d_i}, \forall i \in U. \tag{3.9}$$

From (3.1) and (3.9) we deduce that

$$R_d = \sum_{i \in U} (1/P_{d_i})^{(1/\alpha - 1)}, \alpha \in (0,1). \tag{3.10}$$

When P_{d_i} is smaller than 1, the revenue generated by our pricing methodology decreases as α increases. In contrast, when P_{d_i} is larger than 1, the revenue generated by our pricing methodology increases as α increases.

3.3 Case Study on Knapsack Combinatorial Optimization

Combinatorial optimization optimizes the total value of the items that could fit in a container with fixed volume. Knapsack problem is a popular combinatorial optimization problem widely applied in resource allocation and cost-benefits analysis where there are financial constraints exist [56][57]. Thus, in this section, we study our proposed pricing methodology with knapsack allocation algorithm.

3.3.1 Knapsack Problem

In general knapsack problem, given a set of n items, each item with a weight k_i and value v_i, and also given the maximum weight K of the container, the knapsack problem determines the number of items that can be put into the container with the optimization goal of maximizing the total value of the items. When applying knapsack problem into resource allocation of cloud data center with resource capacity vector c, given T_{s_i} VMs requested by cloud tenant i, each VM with a resource vector D_i and price P_{e_i}, the optimization goal is to maximize the total revenue of a cloud provider given the amount of resources user requested VMs contain, the prices of each VM and the total resource available in the cloud data center.

0-1 Knapsack Problem

0-1 knapsack constraints that each type of the VM requested by user i is allocated on cloud servers with no repetition. The mathematical expression of the 0-1 knapsack problem is,

$$\begin{aligned} \text{maximize} \quad & \sum_{i=1}^{n} P_{e_i} T_{n_i} \\ \text{subject to} \quad & \sum_{i=1}^{n} D_i T_{n_i} \leq c, T_{n_i} \in (0,1). \end{aligned} \tag{3.11}$$

Bounded Knapsack Problem

Bounded knapsack problem allows multiple but limited VMs requested by user i to be allocated on cloud servers. Let T_{s_i} denotes the maximum number of VM each VM type i that user requests, then,

$$\begin{aligned} \text{maximize} \quad & \sum_{i=1}^{n} P_{e_i} T_{n_i} \\ \text{subject to} \quad & \sum_{i=1}^{n} D_i T_{n_i} \leq c, T_{n_i} \in (0, \ldots, T_{s_i}). \end{aligned} \tag{3.12}$$

Unbounded Knapsack Problem

Unbounded knapsack problem allows unlimited VMs requested by user i to be allocated on cloud servers. Thus, the problem is formulated as,

$$\begin{aligned} \text{maximize} \quad & \sum_{i=1}^{n} P_i T_{n_i} \\ \text{subject to} \quad & \sum_{i=1}^{n} D_i T_{n_i} \leq c, T_{n_i} \geq 0. \end{aligned} \tag{3.13}$$

3.3.2 Knapsack-based Revenue Enhancement

Since there are possibilities that different users would require the same VM, or the same user would require multiple VMs of the same type, we study our pricing methodology with bounded knapsack problem. The mathematical formulation of pricing weight is,

$$w_i = \frac{\frac{1}{T_{n_i}}}{\sum_{i \in U} \frac{1}{T_{n_i}}}, \forall i \in U \tag{3.14}$$

Knapsack-based Revenue Enhancement with Total Unit Price Redistribution

The calculation of P_{d_i} is straight forward,

$$P_{d_i} = w_i \times \sum_{i \in U} P_{e_i}, \forall i \in U. \tag{3.15}$$

Knapsack-based Revenue Enhancement with Total Revenue Redistribution

With the number of VM that is allocated for each user *i*, the calculation of P_{d_i} is as follows,

$$R_{e_i} = T_{n_i} \times P_{e_i}, \forall i \in U. \tag{3.16}$$

New unit price $P_d = \{P_{d_1}, \ldots, P_{d_n}\}$ is calculated as

$$P_{d_i} = \frac{w_i \times \sum_{i \in U} R_{e_i}}{x_i}, \forall i \in U. \tag{3.17}$$

3.3.3 Simulations

In order to evaluate the proposed pricing methodology, we conduct experiments to compare the total revenue our pricing model produces with the total revenue a pricing model with flat-rate unit price would produce. We use actual Google Compute Engine VM types to model our cloud system. To be specific, we assume a cloud provider with unified resource formulated as $< CPU, Memory >$, where Memory is represented in terms of gigabyte (GB) and four users each requesting a VM with resource formulated in terms of $< CPU, Memory >$. We compare the revenue of our pricing model with the revenue a cloud provider would receive with Google Compute Engine unit prices. New prices are generated once a request of VM has arrived at the system. Then, user responses to the new prices are modeled according to the demand function described in Section 3.2.1. Finally, the revenue of our pricing model is calculated as the product of the new unit prices and users' final decision of the amount of the VMs they request. Furthermore, we compare the change of user requests with our new unit prices to evaluate the ability of demand control of our proposed pricing methodology. The numeric data used in the simulations is listed in Table 3.2 and Table 3.3 .

Table 3.2: System Resource Vector for Knapsack-based Revenue Enhancement

Total Resource	$< 40CPU, 180GB >$

From Table 3.3 we observe that User A requested VM is small so while it is easy to be allocated

Table 3.3: User Demand Vector and Prices of Google Compute Engine

User A	$< 1CPU, 3.75GB >$	\$ 0.045/hour
User B	$< 4CPU, 3.6GB >$	\$ 0.112/hour
User C	$< 2CPU, 7.5GB >$	\$ 0.089/hour
User D	$< 4CPU, 15GB >$	\$ 0.177/hour

using resource fragments, it also would fragmentize server resources. User B requested VM is CPU intensive. It would quickly use up server CPUs but left too much Memory. User C requested VM is medium sized which is easier to be allocated than User D requested VM but also easier to fragmentize server resources.

Simulation for Knapsack-based Revenue Enhancement with Total Unit Price Redistribution

In Figure 3.3, we compare the revenue our pricing methodology generates with alpha set to 0.5 to 0.9. The revenue decreases as the alpha value increases. The overall revenue is higher comparing to the revenue generated with Google Compute Engine unit prices.

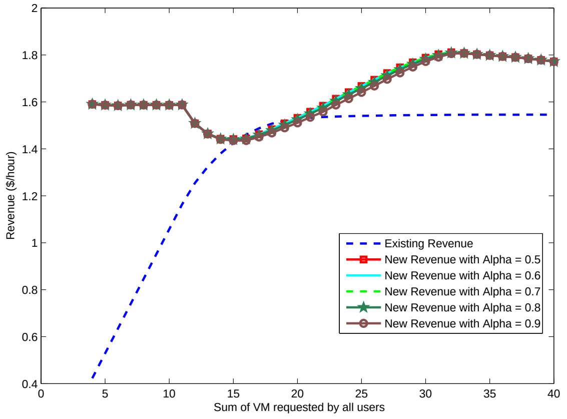

Figure 3.3: Comparison of Revenues with Different Alpha Values

In Table 3.4 we show the new unit prices our pricing methodology produces. The unit price of the VM User A requested is raised by 17.8% and the unit price of User B requested VM is raised by 135.7%

Table 3.4: New Unit Prices Compared To Google Compute Engine Unit Prices

VM Type	Number of VMs Requested	Google Prices	New Prices
$< 1CPU, 3.75GB >$	5	\$ 0.045/hour	\$ 0.053/hour
$< 4CPU, 3.6GB >$	5	\$ 0.112/hour	\$ 0.264/hour
$< 2CPU, 7.5GB >$	5	\$ 0.089/hour	\$ 0.053/hour
$< 4CPU, 15GB >$	5	\$ 0.177/hour	\$ 0.053/hour

to discourage User A and, especially, User B from requesting resources. The unit price of the VM User C requests is lowered by 40.4% while the unit price of the VM User D requested is lowered by 70% to encourage User C and D to request more resources.

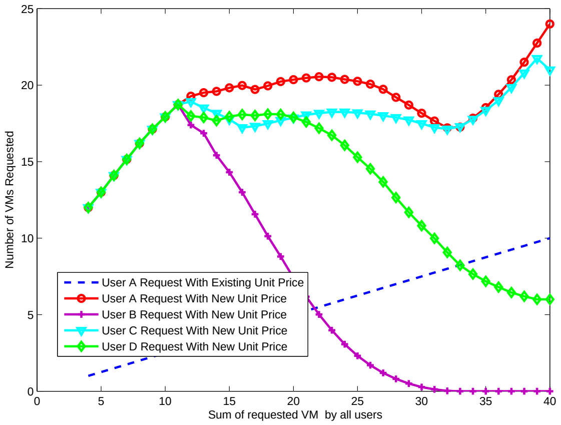

Figure 3.4: Comparison of Requests Generated by Each User

In Figure 3.4, we compare the change of user requests with our new unit prices of the types of VMs used in example with α set to 0.9. To maximize the total revenue and the server utilization level, a cloud provider needs to sell as much VMs as possible. The VM User B requests is the least encouraged according to the knapsack algorithm because the CPU would quickly be used up if User B requests too much. Therefore, from the simulation result, the number of VMs User B requests as a response to our new unit prices is changed to 0 eventually. Also, since the VM User D requested is large, User D is starting to be discouraged to request resources as the server is fully utilized and the sum of requested VM by all users goes to 40. On the other hand, the VM User C requests is the most encouraged according to knapsack algorithm. So the number of VMs User C requests as a response to our new unit prices is increased the most.

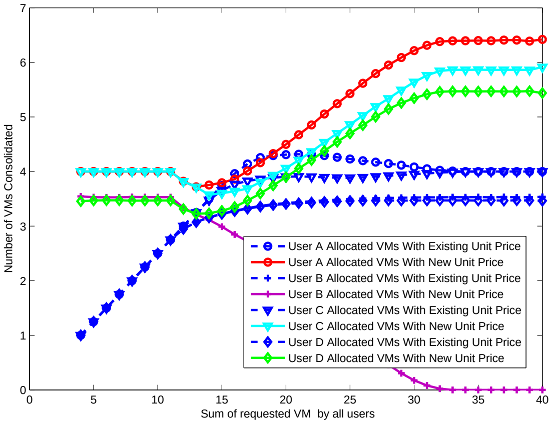

Figure 3.5: Comparison of Allocated Virtual Machines for Each User

In Figure 3.5 and Figure 3.6, we further show the change of allocated VMs for each user and compare the revenue each user generates with our new unit prices. The result show to correctness of our pricing methodology since it indeed controlled user demand in a way that Knapsack Algorithm suggests.

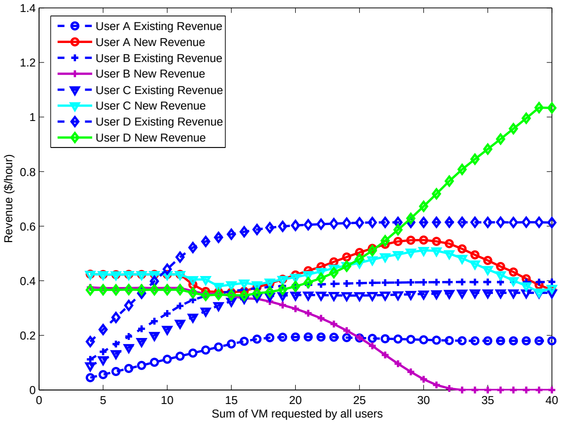

Figure 3.6: Comparison of Revenues Generated by Each User

Simulation for Knapsack-based Revenue Enhancement with Total Revenue Redistribution

In Figure 3.7, we compare the revenue our pricing methodology generates with alpha set to 0.5 to 0.9. The revenue decreases as the alpha value increases. The overall revenue is higher comparing to the revenue generated with Google Compute Engine unit prices.

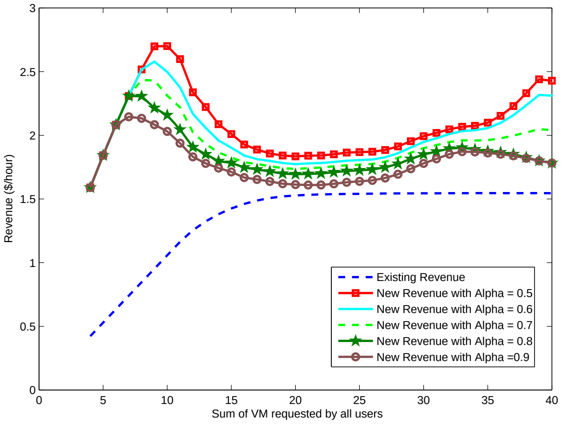

Figure 3.7: Comparison of Revenues with Different Alpha Values

In Table 3.5 we compare the new unit price our pricing methodology produces with the existing unit prices the Google Compute Engine charges. User A, C, and D are encouraged to request more resources

Table 3.5: New Unit Prices Compared To Google Compute Engine Unit Prices

VM Type	Number of VMs Requested	Google Prices	New Prices
$< 1CPU, 3.75GB >$	5	\$ 0.045/hour	\$ 0.042/hour
$< 4CPU, 3.6GB >$	5	\$ 0.112/hour	\$ 1.042/hour
$< 2CPU, 7.5GB >$	5	\$ 0.089/hour	\$ 0.042/hour
$< 4CPU, 15GB >$	5	\$ 0.177/hour	\$ 0.042/hour

since the unit price User A requested is lowered by 6.7%, the unit price User C requested is lowered by 52.8% and the unit price User D requested is lowered by 76.3%. The unit price User B requested is raised by 803.3% to discourage User B from requesting resources.

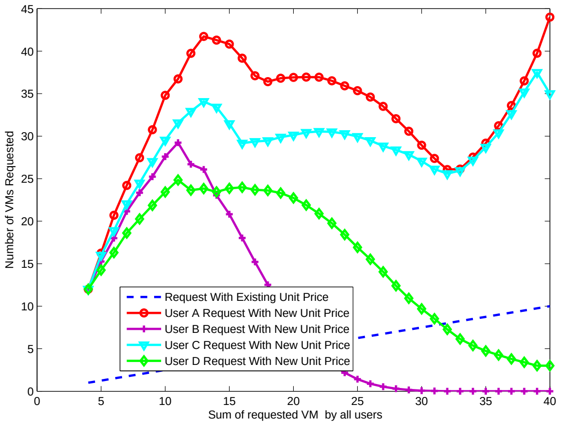

Figure 3.8: Comparison of Requests Generated by Each User

In Figure 3.8, we compare the change of user requests. Same as the simulation results of TUPR, User B is the least encouraged to request VM, so the number of VMs User B requests is gradually changed to 0. User D is discouraged to request VM as the sum requested VM by all users goes to 40. User C is most encouraged to request VM, so the number of VMs user C requests increased the most.

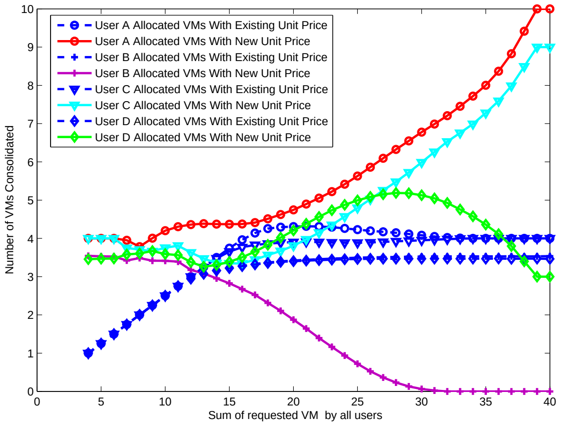

Figure 3.9: Comparison of Allocated Virtual Machines for Each User

In Figure 3.9 and Figure 3.10, we compare the change of allocated Virtual Machines of each user with our new unit prices and the revenue each user generates with our new unit prices. The results shows that our pricing methodology is able to control demand according to the suggestion of Knapsack algorithm.

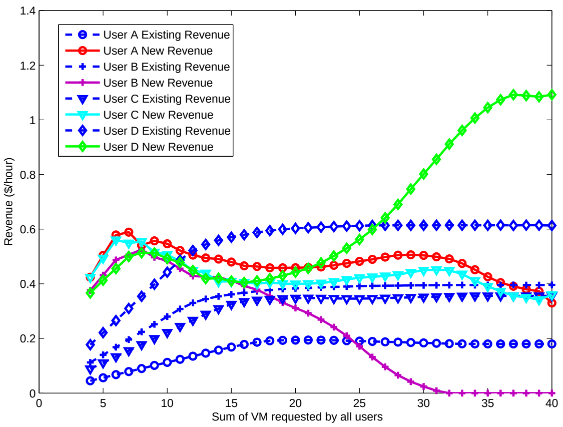

Figure 3.10: Comparison of Revenues Generated by Each User

3.3.4 Discussion of Knapsack-based Revenue Enhancement

The knapsack problem maximizes the total revenue for the cloud provider. However, it does not consider fairness among users. Some of the desirable fairness properties are sharing incentive that guarantees that no user is better off in a system where resources are equally partitioned among all users; strategy-proofness, where no user can be better off by providing untruthful resource demands; Pareto-efficiency, where user requests are allocated without preempting existing allocations; envy-freeness, where no user prefers the allocation of another user [15]. From the simulation results, we can see that eventually User B will not be able to use any of the cloud resources. We claim that it is not fair for User B. However, as discussed in previous chapter, the fairness-efficiency trade-off is also an important factor for any algorithms or methodologies that tries to achieve fairness. How to find an acceptable threshold has become an interesting question. To this end, we study our pricing methodology with the DRFH algorithm, which is proved to be able to achieve fairness and a higher system utilization than several fair schedulers in next section.

3.4 Case Study on DRFH Fairness

In [16], authors have proved that the DRFH allocation algorithm achieves significant improvements in resource utilization, compared to the traditional slot scheduler (e.g., Hadoop Fair Scheduler). Thus, in this section, we study our proposed pricing methodology with DRFH allocation algorithm.

3.4.1 DRFH Scheduling

DRFH takes the set of heterogeneous servers S, server resource types (e.g., CPU, Memory, storage) S_r and normalized resource capacity vector for each server l, c_l, as one input and takes the set of cloud tenants U and resource vector of demanded VM D_i as another input. Let r_i^* be the largest resource required by user i's VM, then the normalized demand of user i's VM is calculated as

$$d_{ir} = D_{ir}/D_{ir_i^*}, \forall i \in U, r \in R. \tag{3.18}$$

For example, consider a system with 2 servers, Server 1 with 2 CPUs, 12GB Memory and Server 2 with 12 CPUs, 2GB Memory. The system has 14 CPUs, 14GB Memory in total. Thus, Server 1 and Server 2 have normalized resource capacity vectors $< 1/7, 6/7 >$ and $< 6/7, 1/7 >$, respectively. Suppose there are two users. User1 requests VMs with 0.1 CPUs and 0.5GB Memory. User2 requests VMs with 1 CPU and 0.2GB Memory. User1 and User2 have normalized demand vectors $< 1/5, 1 >$, $< 1, 1/5 >$, respectively.

Let $A_{il} = \{A_{il1}, \ldots, A_{ilm}\}$ be the resource allocation vector for user i on server l, $A_i = \{A_{i1}, \ldots, A_{ik}\}$ be the allocation matrix of user i and $A = \{A_1, \ldots, A_n\}$ be the overall allocation for all users. An allocation is feasible only if no server has used more resources than its total resources

$$\sum A_{ilr} \leq c_{lr}, \forall l \in S, r \in R. \tag{3.19}$$

Let $N_{il}(A_{il})$ be the maximum number of VMs that can be scheduled for user i in server l and D_{ir} be the fraction of the total resource r required by user i, then,

$$N_{il}(A_{il}) = \min_{r \in R} \{A_{ilr}/D_{ir}\}, \forall l \in S, r \in R. \tag{3.20}$$

Let $N_i(A_i)$ be the total number of VMs that can be scheduled for user i under allocation A_i, then,

$$N_i(A_i) = \sum_{l \in S} N_{il}(A_{il}). \tag{3.21}$$

The dominant resource allocated to user i in server l is

$$G_{il}(A_{il}) = N_{il}(A_{il}) D_{ir_i^*} = \min_{r \in R} \{A_{ilr}/d_{ir}\}. \tag{3.22}$$

Therefore, the global dominant share allocated to user i is

$$G_i(A_i) = \sum_{l \in S} G_{il}(A_{il}) = \sum_{l \in S} \min_{r \in R} \{A_{ilr}/d_{ir}\}. \tag{3.23}$$

The goal of DRFH is to maximize the minimum global dominant share among all users, subject to the capacity constraints of each server,

$$\max_{A} \min_{i \in U} G_i(A_i),$$
$$\text{s.t.} \sum A_{ilr} \leq c_{lr}, \forall l \in S, r \in R. \tag{3.24}$$

DRFH allocates resource to the user with the minimum global dominant share among all active users. When a user has all its VMs placed in a server, it is removed from the user set U and DRFH repeats the allocation process with the updated user set.

3.4.2 DRFH-based Revenue Enhancement

As introduced in Section 3.2.3, the pricing weight calculation for DRFH-based revenue enhancement methodology is

$$w_i = \frac{\frac{1}{N_i(A_i)}}{\sum_{i \in U} \frac{1}{N_i(A_i)}}, \forall i \in U \tag{3.25}$$

DRFH-based Revenue Enhancement with Total Unit Price Redistribution

The calculation of P_{d_i} is

$$P_{d_i} = w_i \times \sum_{i \in U} P_{e_i}, \forall i \in U. \tag{3.26}$$

DRFH-based Revenue Enhancement with Total Revenue Redistribution

Next, the calculation of P_{d_i} for TRR is

$$R_{e_i} = N_i(A_i) \times P_{e_i}, \forall i \in U. \tag{3.27}$$

New unit price $P_d = \{P_{d_1}, \ldots, P_{d_n}\}$ is calculated as

$$P_{d_i} = \frac{w_i \times \sum_{i \in U} R_{e_i}}{N_i(A_i)}, \forall i \in U. \tag{3.28}$$

3.4.3 DRFH-based Fairness

In [15], authors showed that unlike the fair division mechanism Competitive Equilibrium from Equal Incomes [58], which does not satisfy strategy-proofness and population monotonicity, and Asset Fairness, which does not satisfy sharing incentive and bottleneck fairness, DRF satisfies sharing incentive, strategy-proofness, envy-freeness, Pareto-efficiency, single resource fairness, bottleneck fairness and population monotonicity at the mean time. In [16], authors proved by deduction that DRFH, as an extension of DRF, satisfies strategy-proofness, envy-freeness, Pareto-efficiency, single resource fairness, bottleneck fairness and population monotonicity as the DRF does. Furthermore, authors in [16] showed through the trace-driven evaluation that although DRFH does not guarantee 100% sharing incentive for all users, it provides 98% of users the same task completion ratio as traditional Slots schedulers do. Since the calculation of our new prices is inversely proportional to the number of tasks DRFH algorithm can schedule for a user, our proposed pricing methodology satisfies the same fairness properties as DRFH. That is, our proposed pricing methodology provides sharing incentive for most of the users; it is strategy-proof, envy-free and Pareto-efficient, and satisfies single resource fairness, bottleneck fairness and population monotonicity.

For the same example described previously, suppose a cloud provider charges for one VM with 0.1 CPU and 0.5GB RAM at $0.6/hour and one VM with 1 CPU and 0.2 GB RAM at $1.2/hour. Also assume each user requests an infinite number of VMs. DRFH allocates 12 VMs to user1 and 6 VMs to user2. The new prices are calculated as $0.4/hour and $1.6/hour for user1 and user2, respectively.

Personal Fairness Since by Personal Fairness [59], most users would expect that a VM consisting of smaller resources is cheaper than a VM that consists of larger resources, the new prices of $0.4/hour for a 0.1 CPU and 0.5GB RAM VM and $1.6/hour for a 1 CPU and 0.2 GB RAM VM indicate personal fairness of our pricing model.

Social Fairness Social fairness means that the provider does not profit unreasonably and the price only increases based on the increase of costs. Since server capacity is limited, one VM with a larger amount of resources allocated in a server could use up a large portion of the server's resources and the remaining resources might not be sufficient for allocation to other users' VMs. Therefore, a VM with larger amount of resources is more costly to allocate. Our new prices are inversely proportional to the number of VMs a user receives. With the goal of DRFH allocation algorithm to equalize the share of resources among all active users, fewer VMs allocated indicates the type of VM requested consists of larger amount of resources. VM with larger amount of resources is more costly to allocate, so it should be charged more. Comparing to the original prices, the decrease in user1's price and increase in user2's price indicate social fairness of our pricing model.

3.4.4 Simulations

In order to evaluate the proposed fairness-aware pricing methodology, we conduct experiments to compare the total revenue our pricing model produces with the total revenue a pricing model with flat-rate unit price would produce. To be specific, in numerical evaluations, we use actual Google Compute Engine VM types to model our cloud system. We compare the revenue of our pricing model with the revenue a cloud provider would receive with Google Compute Engine unit prices. In the trace-driven evaluation, we use the sum of CPU and memory of each requesting VM as the unit price of this VM. Then, we compare the revenue of our pricing model with the revenue a cloud provider would receive with the calculated unit prices. New prices are generated once a request of VM has arrived at the system. Then, user responds to the new prices are modeled according to the demand function described in Section 3.2.1. Finally, the revenue of our pricing model is calculated as the product of the new unit prices and users' final decision of the amount of the VMs they request.

Numerical Simulations

In this section, we demonstrate the efficiency of our algorithm with numerical simulations with 2 types of computing resources, including CPU units and Memory. A cloud service provider with 2 servers is hosting 4 users' applications simultaneously. We formulate the servers' resource as a vector $< CPU, Memory >$, represented in GB, and all users' request as a resource vector $< CPU, Memory >$ and corresponding unit prices from Google Compute Engine. These numeric data are listed in Table 3.6 and Table 3.7, respectively. We use FIFO scheduling to simulate the process of VM allocation and at the end of scheduling process, get the actual number of VMs each user could receive.

Table 3.6: System Resource Vector

Server 1	$< 27CPU, 50GB >$
Server 2	$< 13CPU, 50GB >$
Total Resource	$< 40CPU, 100GB >$

Table 3.7: User Demand Vector and Prices of Google Compute Engine

User A	$< 1CPU, 3.75GB >$	$ 0.07/hour
User B	$< 4CPU, 3.6GB >$	$ 0.176/hour
User C	$< 2CPU, 7.5GB >$	$ 0.14/hour
User D	$< 4CPU, 15GB >$	$ 0.28/hour

DRFH-based Revenue Enhancement with Total Unit Price Redistribution

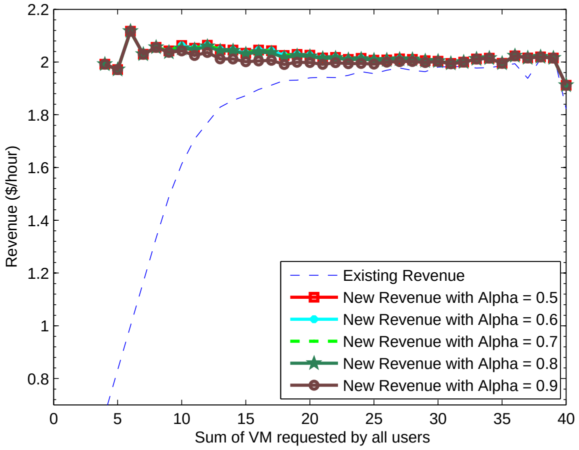

Figure 3.11: Comparison of Revenues Generated by Existing and Proposed Pricing Model

Figure 3.11 illustrates the comparison of revenues achieved with unit prices of Google Compute Engine and proposed policies with α set to 0.5, 0,6, 0.7, 0.8 and 0.9. Revenues converge as the system resource is fully occupied at which the sum of VM requested by all users is approximately 15. Since the unit prices are smaller than 1, the revenue with a smaller α value is greater than the revenue with a larger α value, as depicted in Section 3.2.1. Overall, new revenue generated by our proposed pricing methodology is higher than the existing revenue calculated with unit prices of Google Compute Engine.

Table 3.8: New Unit Prices Compared to Google Compute Engine Unit Prices

VM Type	Number of VMs Requested	Google Prices	New Prices
$< 1CPU, 3.75GB >$	5	\$ 0.07/hour	\$ 0.097/hour
$< 4CPU, 3.6GB >$	5	\$ 0.176/hour	\$ 0.162/hour
$< 2CPU, 7.5GB >$	5	\$ 0.14/hour	\$ 0.162/hour
$< 4CPU, 15GB >$	5	\$ 0.28/hour	\$ 0.244/hour

In Table 3.8 , we compare our new unit prices of each user with Google Compute Engine unit prices for the types of VMs used in the example with each user requesting 5 VMs of their desired type. The unit price of the VM User A requested is raise by 38.6% and the unit price of VM User C requested is raised by 15.7%, where as the unit price of VM User B requested is lowered by 8.0% and the unit price of VM User D requested is lowered by 12.9%. The VM with $< 1CPU, 3.75GB >$ that User A requested requires the smallest portion of resources so that it is easy to be placed on a server. Therefore, the unit price of VM $< 1CPU, 3.75GB >$ is the lowest. On the other hand, the VM with $< 4CPU, 15GB >$ that User D requires the largest portions of resources so that it is hard to be placed on a server. Hence, the new price of this type of VM is the highest.

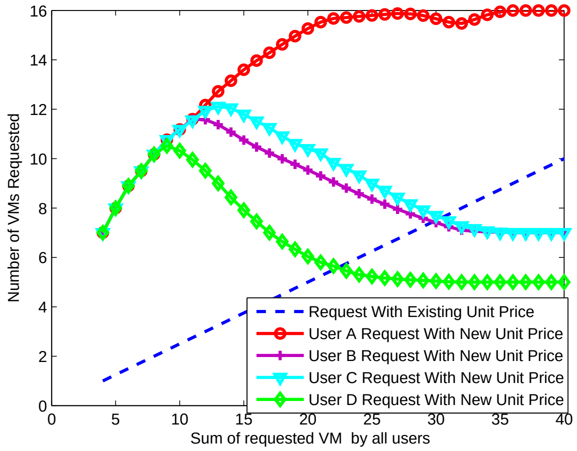

Figure 3.12: Comparison of Requests Generated by Each User

In Figure 3.12, we compare the change of user requests of our new unit prices of the types of VMs used in the example with α set to 0.9. The VM with $< 1CPU, 3.75GB >$ is charged at the lowest price; therefore, the number of VMs User A requests is increased the most. On the other hand, the VM with $< 4CPU, 15GB >$ is charged at the highest price; therefore, as a respond to the price increases, User D requests less and less VMs.

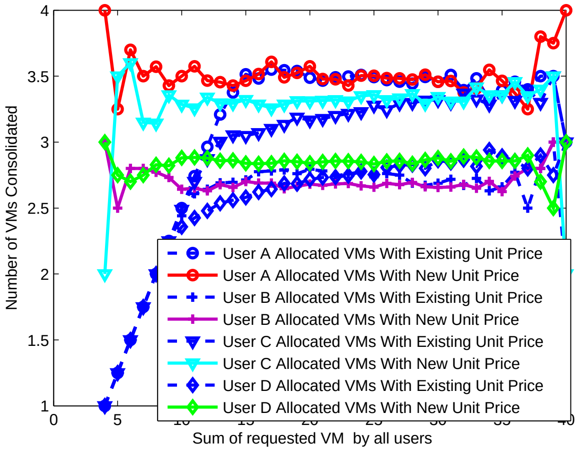

Figure 3.13: Comparison of Allocated Virtual Machines for Each User

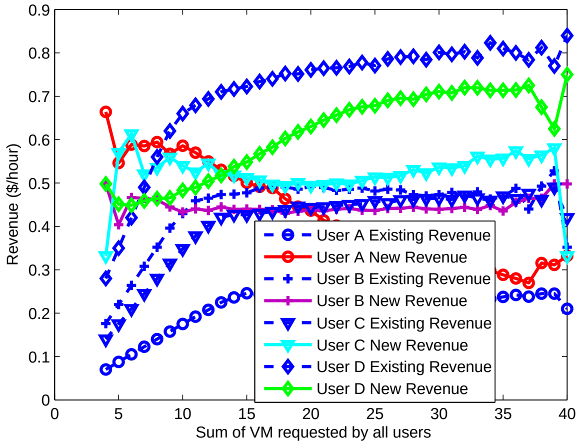

Figure 3.14: Comparison of Revenues Generated by Each User

In Figure 3.13, we show the number of VMs allocated on a server for each user with our new unit prices with α set to 0.9. In Figure 3.14, we compare the revenue generated by each user with our new unit prices of the types of VMs used in the example with α set to 0.9. Compared to the revenue achieved with Google Compute Engine unit prices, the results show that revenues generated with our pricing methodology by users B and D are decreased, whereas the revenues generated by users A and C are increased. Still, with our pricing methodology, the greatest revenue is generated by user D as it requires the type of VM that is most difficult to handle; the next greatest revenue is generated by user C, and then user B and user A, which indicates personal and social fairness.

DRFH-based Revenue Enhancement with Total Revenue Redistribution

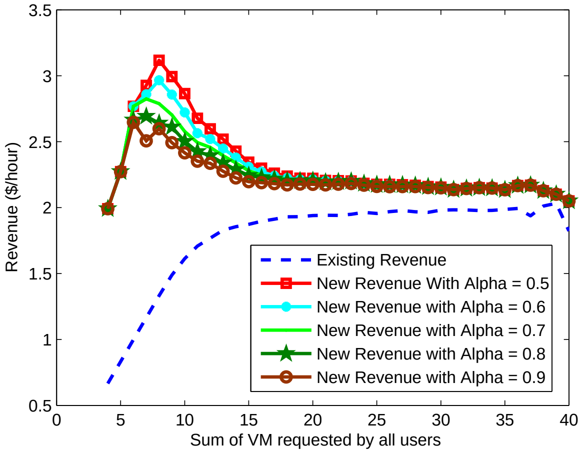

Figure 3.15: Comparison of Revenues Generated by Existing and Proposed Pricing Model

Figure 3.15 illustrates the comparison of revenues achieved with unit prices of Google Compute Engine and proposed policies with α set from 0.5 to 0.9. Revenues converge as the system resource is fully occupied. Again, the revenue with $\alpha = 0.5$ is greater than the revenue with $\alpha = 0.9$. Overall, new revenue generated by our pricing methodology is higher than the revenue calculated with the existing unit prices of Google Compute Engine.

Table 3.9: New Unit Prices Compared To Google Compute Engine Unit Prices

VM Type	Number of VMs	Google Prices	New Prices
$< 1CPU, 3.75GB >$	5	$ 0.07/hour	$ 0.054/hour
$< 4CPU, 3.6GB >$	5	$ 0.176/hour	$ 0.151/hour
$< 2CPU, 7.5GB >$	5	$ 0.14/hour	$ 0.151/hour
$< 4CPU, 15GB >$	5	$ 0.28/hour	$ 0.34/hour

In Table 3.9, we compare the unit prices generated for each user's requested VM type with the unit prices provided by Google Compute Engine. The unit price of VM User A requested is lowered by

22.9% and the unit price of VM User B requested is lowered by 14.2%, where as the unit price of VM User C requested is raised by 7.9% and the unit price of VM User D requested is raised by 21.4%.

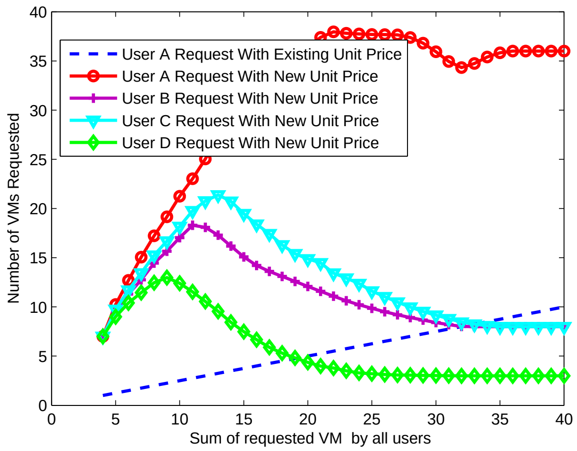

Figure 3.16: Comparison of Requests Generated by Each User

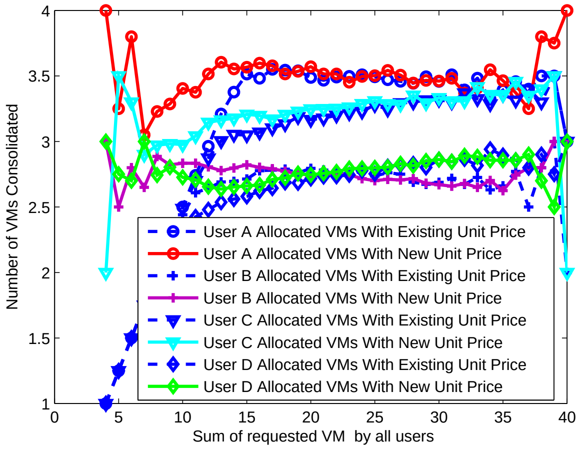

Figure 3.17: Comparison of Allocated Virtual Machines for Each User

In Figure 3.16, we compare the requests submitted to the cloud system by each user with the existing and new unit prices. All users are able to request for resource. As the sum of requested VM by all users goes to 40, the number of VMs User A requests is increased the most and the number of VMs User D requests is decreased the most.

In Figure 3.17 we show the differences of the number of requested VMs allocated for each user with the existing and new unit prices. In Figure 3.18, we compare the revenue generated by each user with α set to 0.9. The results show that, compared to the revenue achieved with Google Compute Engine unit prices, the revenues generated by users A and B are decreased with our new unit prices, whereas revenues generated by users C and D are increased with our new unit prices. Furthermore, the greatest revenue is generated by user D as it requires the type of VM that is most difficult to handle; the next greatest revenue is generated by user C, and then user B and user A, which indicates personal and social fairness.

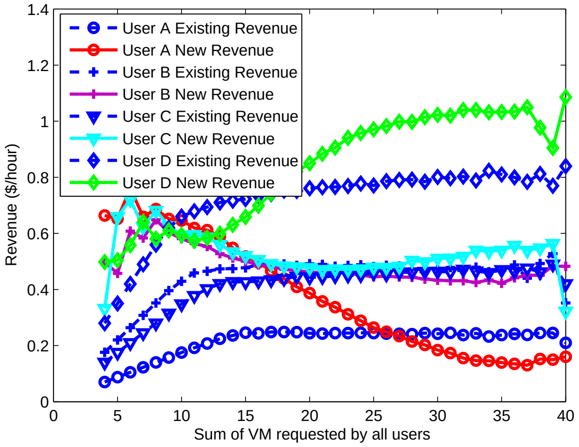

Figure 3.18: Comparison of Revenues Generated by Each User

Trace-driven Simulations

In this section, we evaluate our proposed pricing methodology with empirical data from Google cluster-usage traces[8]. The traces contain user resource requests and system resource information for about a month-long period in May 2011. We extract the system resource vector (S and S_r), set of active cloud tenants (U), resource vector of the VM requested by the ith user (d_i), and the number of VM requests submitted (T_{si}) from the traces. Since server resources provided in Google cluster-usage traces are normalized to the largest server resource, the actual Google Compute Engine price of the requested type

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

of VM is hard to determine. Thus, we assume that originally the cloud provider charges \$1 per CPU per hour and \$1 per GB Memory per hour, where both CPU and Memory are normalized numbers extracted from Google cluster-usage traces. For example, a server with normalized 0.5 CPU and 1 GB Memory is originally charged at \$1.5 per hour.

As described in Section 3.2.1, elasticity coefficient α is the decisive parameter for market reaction. However, the values of α are different from user to user, subject to distinct application domain and various motivations. To simulate a reasonable scenario, the elasticity coefficient α is derived by dynamic calculations with price and demand settings in our experiments. Specifically, the value of α is calculated by the following equation for our trace-driven simulations:

$$\alpha = \frac{1}{\log_{1/p} f_R(p)}. \tag{3.29}$$

DRFH-based Revenue Enhancement with Total Unit Price Redistribution

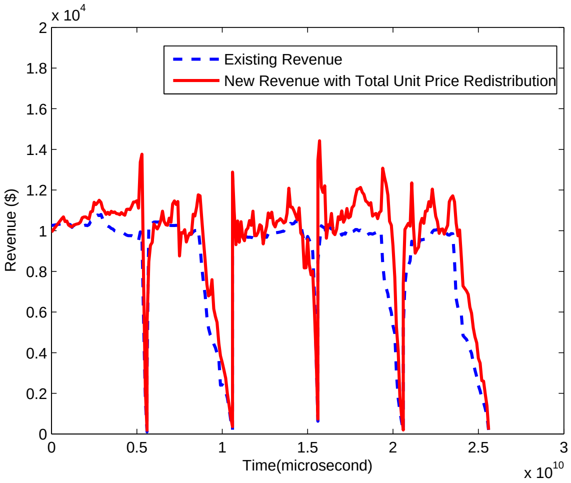

Figure 3.19: Comparison Between Existing and Proposed Pricing Model

The trace-driven simulation results for total unit price redistribution are depicted in Figure 3.19. The revenue of the cloud provider is calculated according the user demands extracted from the traces and the assumed original unit prices from Google Compute Engine and unit prices generated by our proposed pricing methodology. The user demands vary throughout the monitoring period, so does the cloud provider's revenue, which ranges from around \$0 to around \$15000. From the results we observe that our proposal outperforms existing pricing policy in most of the time slots. Cumulatively, the cloud service providers' overall revenue can be increased from \$2173700 to \$2426000, which is by up to 11.60%.

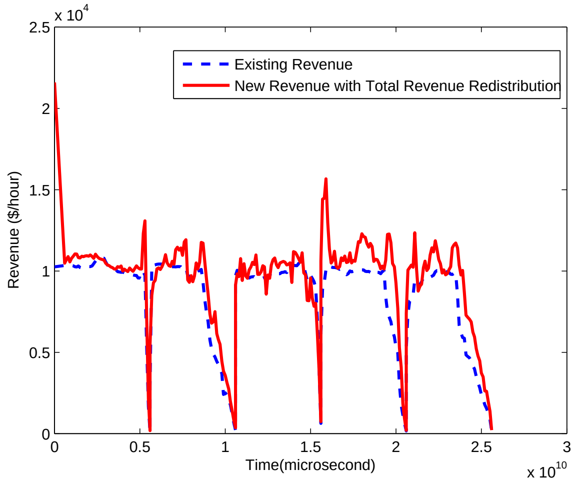

Figure 3.20: Comparison Between Existing and Proposed Pricing Model

DRFH-based Revenue Enhancement with Total Revenue Redistribution

In contrast, we also conducted evaluations for the pricing of total revenue redistribution. From the results shown in Figure 3.20, we observe that the cloud service provider derives more revenue with the novel pricing solution, comparing to existing flat pricing approach. The overall revenue increased significantly from \$2172800 to \$2415600, which is around 11.18%.

3.5 Summary

In this chapter, we have proposed a pricing methodology that induces the optimal resource demand pattern and enhances the revenue of cloud providers. New prices are determined according to the number of tasks a user could get scheduled by the cloud task scheduling algorithm. A user whose task is difficult for the system to process is discouraged from being submitted to the system as frequently as other users' tasks that can be handled easily. Our new prices are calculated with TUPR and TRR. Then, we have studied our methodology with the Knapsack Algorithm and DRFH allocation algorithm. Numerical simulations have been conducted to compare the total revenue a cloud provider would receive with the new unit prices generated by our pricing methodology in both of the cases to the total revenue a cloud provider would receive with its original prices. Simulation results have also shown that our pricing methodology is able to control demand according to the allocation algorithms' suggestions. An empirical study with trace-driven simulation results has shown that the proposed pricing policy with DRFH algorithm can increase the cloud service providers' overall revenue by up to 11.60% and 11.18%, respectively.

Chapter 4

A Price-Aware Cloud Federation System

4.1 The Price-Aware Cloud Federation System

Our pricing methodology proposed in Chapter 3 requires users to constantly pay attentions to the current prices the cloud providers charge. This may be a shortcoming for our pricing methodology to be acceptable or useful. Furthermore, since the cloud is the best effort service, many organizations are afraid to rely on a third-party cloud service to ensure the organizations' service availability and business continuity. Another concern relates to data security, confidentiality and auditability [60]. Crackers and hackers have never stopped their efforts on attacking information systems to steal users' virtual properties. Since the access is public, cloud exposes their systems to more public attacks than conventional data centers. On the other hand, users cannot easily extract their data and programs from one cloud service to another. When a cloud data center crushes, users can hardly move their data to another data center on time. Plenty of research works are devoted to investigate secured and robust cloud service infrastructures. All of these obstacles can be removed by using services from a cloud federation or inter-cloud [8] system that supports distributed cloud services where one cloud could use the computational, storage or network resources of other clouds [61] to complete a task together [62]. Similar to cloud-RAID [48], better privacy protection can be achieved if users were able to control on which clouds the highly sensitive software classes or objects are hosted and distribute the rest of system function codes on other clouds. This way, no public cloud provider could acquire all the program pieces. With programs and data scattered among and backed-up by different data centers maintained by different cloud providers, also with the automated service choosing and program/data sets migration ability, the cloud service and our proposed pricing methodology are more reliable and convincible for users to adopt.

As discussed in Chapter 1, in this chapter, we propose a price-aware cloud federation system intended to provide a tool for users to automatically choose and migrate their applications to a cloud provider that is charging at a more acceptable rate. Further, we take the advantage of program decomposition techniques, where a large program system can be broken down into system functions as smaller classes or objects and data entities[9]. In this chapter, we assume user applications can be decomposed into program and data sets, each set can run on a VM as a task. Our cloud federation system allows users to configure the privacy constraints of their program by defining the number of program or data sets that could be hold in one cloud service concurrently. The system looks up current prices a cloud provider charges for their VM at a pre-defined time interval. Then, each of the program and data sets can be assigned to a cloud service charging at the lowest rate according to the privacy constraints. The cost of running the program or data set at the destination service, along with the cost of migrating the program or data sets to the destination service are considered before the migration decision is made. As a price-aware cognitive system, the proposed platform collects pricing information from multiple clouds and dynamically adapts its execution modality on purpose of minimizing total cost. This dynamic adaption seeks optimal assignment of codes and data sets of user decomposed application, which is a procedure that uses the update-to-date prices to predict future trend of cost and eventually lead to a lower priced solution. Hence, it is also a user-oriented system that benefits its customers financially.

4.1.1 System Overview

Architecture

The architectural framework of the cloud federation system is shown in Figure 4.1. The platform concatenates multiple cloud services to provide a unified inter-cloud environment. Essentially, it is a three-layer software system: application layer (consists of the code layer and the data set layer), platform layer and cloud infrastructure layer, from top to bottom. As the middle-ware between the cloud infrastructure and application layer, the cloud federation platform monitors the real-time pricing from different providers and cognitively adjust the decomposed program and data sets of user applications to minimize the overall price.

[9] https://en.wikipedia.org/wiki/Decomposition

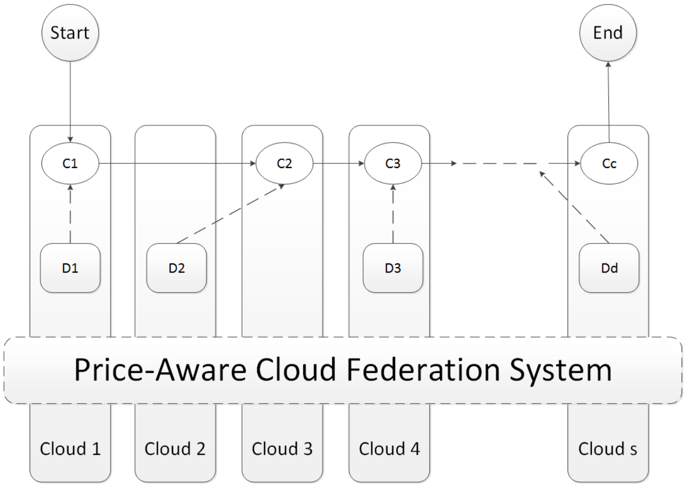

Figure 4.1: Architectural Framework for Cognitive Cloud Federation System

Code Definition

We define c program codes in set C and d data chunks in set D as illustrated as ovals and rectangles in Figure 4.1. We also define s cloud service providers S. Note that some cloud server hosts both code and data set, while some of the others only supports either code or data set. Considering the code-data relationship between program codes and data chunks, we define that a program code can be associated with 0 to d data chunks, while a data chunk can be accessed by 0 to c program codes. Furthermore, we consider the code-data relationship as a directed graph G, where the directed edges are the data flow from data chunks to the program codes, the weights of the edges are the data size to be transmitted. In general, a program code only require a small proportion of data in the data chunk. In this chapter, we denote ω as the data access proportion. Note that if a program code and its associated data chunk are located in different cloud server, an inter-cloud data transmission is required.

Code Migration

One of the key features of our inter-cloud system is the capability of program and data set migrations among multiple cloud services on demand. A mobile agent is a composition of computer software and data that is able to migrate from one cloud to another autonomously and continue its execution at the destination [63][64]. In our cloud federation system, once a better task and cloud assignment solution is found, designated program codes or data chunks can be encapsulated into mobile agents and dispatched to the destination cloud.

Inter-Cloud Message Exchange

In order to facilitate the work flow of user task, the program codes need to communicate with each other through messages, including native context states, processed data and control signals. Since the program codes are executed in a distributed manner among multiple clouds and a single code can be hosted in different clouds under different circumstances, a message exchanges between two code sets can be either local invocations (e.g., when the two code sets are hosted in the same cloud) or remote calls (e.g., when the two code sets are executed in different clouds). Hence, a dynamic message forwarding mechanism that determines the source and destination of a message is needed in the cognitive cloud federation system.

Platform Design

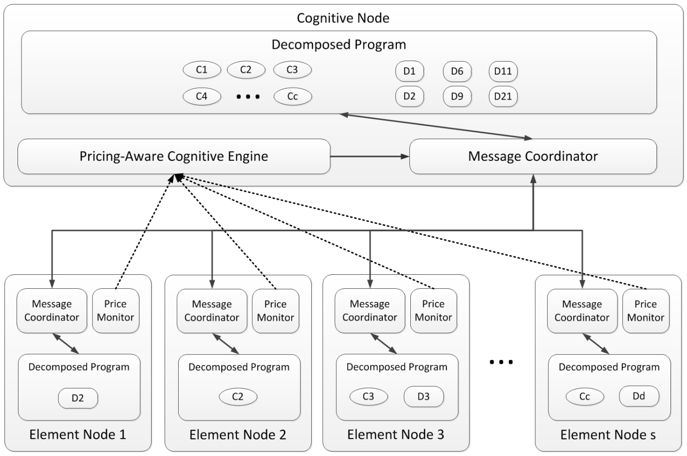

Figure 4.2: Design of Decomposed Cloud Federation Platform

The cloud federation system introduces challenges including decomposition granularity, response latency, synchronization frequency and application programming interface design. In order to facilitate the proposed architectural framework, we design the cognitive cloud federation platform with both privacy and pricing awareness as shown in Figure 4.2.

Similar to the concept of *Master Node* and *Slave Node* implemented in the Hadoop[10] system, the proposed platform incorporates *Cognitive Node* and *Element Node* to facilitate the cognitive feature. A *Cognitive Node* is the core of the whole network, which is hosted in a secured cloud infrastructure, e.g., a private cloud. The *Price-Aware Cognitive Engine* collects the pricing data from *Element Node*s in pre-defined time interval and makes cognitive decisions to optimize the program/data set and the cloud assignment. Note that both *Cognitive Node* and *Element Node* contain a *Message Coordinator*, which serves as the router for inter-code message exchange. The *Price-Aware Cognitive Engine*'s decision provides message routing information, which is broadcasted by the *Cognitive Node* to all *Element Node*s spreading in multiple clouds.

4.2 Problem Formulation

In this section, we mathematically formulate the cloud federation system. For a particular user application, we denote the number of available cloud service providers as cs, the number of program codes as pc, and the number of data chunks as dc.

4.2.1 Program Feature

We model the code-data relationship as a $1-n, \{n = 0, 1, 2, 3.., dc\}$ pair, which means a program code can access 0 to dc data chunks distributed in various cloud servers. We formulate the code-data relationship as $pc \times dc$ logical matrix L, in which the numeric value of elements are defined to be either 0 or 1. Hence, $L_{ij} = 1$ indicates that the ith program code requires data chunk j for its procedure. In addition, we denote the message exchange between code as a $pc \times pc$ matrix M, where M_{ij} indicates the message data size between the ith and the jth program codes.

4.2.2 Dynamic Pricing

With the assumption that each cloud service is charging with dynamic prices, we denote instance prices for the cs cloud providers in vectors P_α and P_β with length of cs, where P_α represents the unit com-

[10] http://hadoop.apache.org/

putational resource prices and P_β represents the unit data storage prices, respectively. In addition, we denote the network prices for the cs cloud providers as P_ρ and P_θ, representing the unit input and output bandwidth prices.

4.2.3 Assignment of Codes and Data Sets

A key of the system design is to determine the assignment of program codes and data chunks. We formulate the assignment of program codes over cloud services as a $pc \times cs$ matrix B_p. It is defined as a logical matrix, where $B_{p_{ij}} = 1$ represents that the ith code is executed at the jth cloud service. Similar to code assignment, we define a $dc \times cs$ logical matrix B_d to formulate the assignment of data chunks over cloud services. $B_{d_{ij}} = 1$ indicates that the ith data chunk is stored at the jth cloud service.

4.2.4 Assignment Constraints

Software Integrity

In order to guarantee the completeness of the software system, every program code and data set chunk is required to be assigned to either one or more cloud services. In order to simplify our model, we only consider the case that no duplicate exists in this work. Therefore, the constraints on software integrity can be described by the following equations:

$$\sum_{j=1}^{n} B_{p_{ij}} = 1, \forall i \in B_{p_{ij}} \tag{4.1}$$

where $B_{p_{ij}}$ represents whether the ith code is executed at the jth cloud service.

$$\sum_{j=1}^{n} B_{d_{ij}} = 1, \forall i \in B_{d_{ij}} \tag{4.2}$$

where $B_{d_{ij}}$ represents whether the ith data chunk is stored at the jth cloud service.

Privacy Assurance

Privacy assurance involves a series of techniques and SLAs, which constrain the access of sensitive data. With strict privacy requirements, some sensitive data even need to be stored in cloud servers within certain geographical areas. For instance, some health care data cannot be transferred outside the US territory, according to specific laws and legislations. In this chapter, we demonstrate the privacy of a software system by a set of *privacy levels* $Theta$ and *Phi*, which represent the privacy restriction on the assignment of program codes and data chunks, respectively. The value of *privacy level* $Theta$, $\Theta \in [1,c]$, is defined as the maximum quantity of program codes allowed to be hosted in the same cloud. Similarly, *Phi*, $\Phi \in [1,d]$, constrains the coexistence of multiple data sets for a specific cloud service. In either of Θ or Φ, a value 1 represents that all program codes or data chunks shall be distributed among different clouds, while the values of pc and dc provide complete freedom for assigning the program codes and data chunks. Thus, the smaller Θ and Φ are, the higher privacy level is assured. With these definitions, we derive the constraints of privacy assurance as follows:

$$\sum_{i=1}^{n} B_{p_{ij}} \leq \Theta, \forall j \in B_{p_{ij}} \tag{4.3}$$

where $B_{p_{ij}}$ represents whether the ith code is executed at the jth cloud service.

$$\sum_{i=1}^{n} B_{d_{ij}} \leq \Phi, \forall j \in B_{d_{ij}} \tag{4.4}$$

where $B_{d_{ij}}$ represents whether the ith data chunk is stored at the jth cloud service.

Apparently, privacy and pricing represent a pair of trade-off that the software users shall be aware of.

4.2.5 Price Calculation

Computing Price

According to above formulations, the total computing price P_c is derived as follows:

$$P_c = F_p^T B_p P_\alpha \tag{4.5}$$

where F_p^T is the transpose of the $pc \times 1$ program code set F_p, B_p is the $pc \times cs$ matrix representing the assignments program codes and cloud services and P_α is the $cs \times 1$ vector of unit computational resource prices.

Storage Price

According to above formulations, the total storage price P_s is derived as follows:

$$P_s = F_d^T B_d P_\beta \tag{4.6}$$

where F_d^T is the transpose of the $dc \times 1$ data chunk set F_d, B_d is the $dc \times cs$ matrix representing the assignments data chunks and cloud services and P_β is the $cs \times 1$ vector of unit data storage prices.

Networking Price

With the data access proportion ω_d, we first derive the $cs \times cs$ data networking volume matrix V_r in following algorithm:

Algorithm 3 Data Networking Matrix Algorithm

```
Initiate a $cs \times cs$ all 0 matrix $V_r$
for each $(i, j)$ in $L_{ij} == 1$ do
    for each $(x, y)$ satisfies $B_{p_{ix}} == 1$ and $B_{d_{jy}} == 1$ do
        if x!=y then
            $V_{r_{yx}} \leftarrow \omega_d F_{d_y}$
        end if
    end for
end for
```

According to the above formulations, the total inbound bandwidth price P_i is derived as follows:

$$P_i = ||V_r P_\rho||_1 \tag{4.7}$$

where V_r is a $cs \times cs$ matrix representing the networking volume between cs cloud services when a piece of data is read from one cloud service and P_ρ is the $cs \times 1$ vector of unit input bandwidth prices. $||.||_1$ is the l_1 norm.

while the total outbound bandwidth price P_o is derived as follows:

$$P_o = ||V_r^T P_\theta||_1 \tag{4.8}$$

where V_r^T is the transpose of the $cs \times cs$ matrix representing the networking volume between cs cloud services when a piece of data is read from one cloud service and P_θ is the $cs \times 1$ vector of unit output bandwidth prices. $||.||_1$ is the l_1 norm.

Also, we derive the $cs \times cs$ message networking volume matrix V_m as follows:

Algorithm 4 Message Networking Matrix Algorithm

```
Initiate a $cs \times cs$ all 0 matrix $V_m$
for each $(i, j)$ in $M_{ij}! = 0$ do
    for each $(x, y)$ satisfies $B_{p_{ix}} == 1$ and $B_{p_{jy}} == 1$ do
        if x!=y then
            $V_{m_{xy}} \leftarrow M_{ij}$
        end if
    end for
end for
```

According to the above formulations, the total message inbound bandwidth price Q_i is derived as follows:

$$Q_i = ||V_m P_\rho||_1 \tag{4.9}$$

where V_m is a $cs \times cs$ matrix representing the networking volume between cs cloud services in case of inter-cloud message exchange and P_ρ is the $cs \times 1$ vector of unit input bandwidth prices. $||.||_1$ is the l_1 norm.

while the total message outbound bandwidth price Q_o is derived as follows:

$$Q_o = ||V_m^T P_\theta||_1 \tag{4.10}$$

where V_m^T is the transpose of the $cs \times cs$ matrix representing the networking volume between cs cloud services in case of inter-cloud message exchange and P_θ is the $cs \times 1$ vector of unit output bandwidth prices. $||.||_1$ is the l_1 norm.

Therefore, the overall network price P_n is derived as

$$P_n = P_i + P_o + Q_i + Q_o \tag{4.11}$$

Total Price

Hence, we derive the total price P of the cloud federation system:

$$P = P_c + P_s + P_n \tag{4.12}$$

4.2.6 State-Transition Cost

When the price changes, the cognitive system needs to adapt to the new optimal solution. However, the transition between two assignment states involves network transmissions. Since the package size of a program code is relatively insignificant comparing to the size of a data chunk, we only consider the networking volume produced by the migration of data chunks. Given the current $dc \times cs$ data chunk assignment matrix B_d at the total price of P, we assume that with an up-to-date pricing status, the proposed system optimizes and derives a set of new $dc \times cs$ data chunk assignment matrix B'_d with new total price P'. The network volume of state-transition $dc \times cs$ matrix V_s can be derived by the following equations:

$$V_s = B'_d - B_d \tag{4.13}$$

Here we derive the inbound matrix I, a $d \times s$ logical matrix, by

$$I_{ij} = \begin{cases} V_{s_{ij}}, & V_{s_{ij}} > 0 \\ 0, & otherwise \end{cases} \tag{4.14}$$

where $V_{s_{ij}}$ represents the ith data chunk state transition cost.

According to the calculation procedure, a non-zero element I_{ij} indicates a network input of the ith data chunk to the jth cloud in the state-transition from B_d to B'_d.

Similarly, we derive the outbound matrix O, a $d \times s$ logical matrix, by

$$O_{ij} = \begin{cases} -V_{s_{ij}}, & -V_{s_{ij}} > 0 \\ 0, & otherwise \end{cases} \tag{4.15}$$

where O_{ij} indicates a network output of the ith data chunk to the jth cloud in the state-transition from B_d to B'_d

Hence, the total inbound bandwidth price R_i is derived as follows:

$$R_i = F_d^T I P_\rho \tag{4.16}$$

where F_d is the transpose of the $dc \times 1$ data chunk set F_d, I is the $dc \times cs$ matrix representing the inbound network volume and P_ρ is the $cs \times 1$ vector of input bandwidth prices.

while the total outbound bandwidth price R_o is derived as follows:

$$R_o = F_d^T O P_\theta \tag{4.17}$$

where F_d^T is the transpose of the $dc \times 1$ data chunk set F_d, O is the $dc \times cs$ matrix representing the outbound network volume and P_θ is the $cs \times 1$ vector of output bandwidth prices.

Therefore, the state-transition pricing R can be derived by:

$$R = R_i + R_o \tag{4.18}$$

4.2.7 Optimization Target

In the starting stage of the system, the price-aware feature requires the system to seek an optimal combination of program code assignments F_p and data chunk assignments F_d that minimizes the overall cost according to a given prices. Hence, the optimization target is formulated as follows:

$$\begin{aligned} \text{Minimize:} \quad & P(F_p, F_d) \\ \text{Subject to:} \quad & (4.1)(4.2)(4.3)(4.4) \end{aligned} \tag{4.19}$$

Once the system is launched, to maintain the price-aware performance, the cognitive engine needs to make decisions to adjust the assignments of program codes and data chunks, according to the real-time pricing fluctuation. It is a continuous process that keeps the assignment up-to-date with dynamic prices. Note that, this procedure does not only simply seeking for the lowest prices, but also need to take the state-transition cost into consideration. With the assumption that pricing policy for all clouds will last for a time interval t, the system is able to make decisions by calculating the overall cost with current price P and new minimal price P' with the state-transition price R. Hence, the objective function is constructed as:

$$\begin{aligned} \text{Minimize:} \quad & R + P'(F_p, F_d)t \\ \text{Subject to:} \quad & R + P'(F_p, F_d)t < P(F_p, F_d)t \\ & (4.1)(4.2)(4.3)(4.4) \end{aligned} \tag{4.20}$$

4.3 Simulations

To validate the performance of our proposed system and the efficiency of optimization approach, we conduct simulations from the perspectives of pricing optimization and cognitive price-aware transition.

4.3.1 Simulation Setup

Table 4.1: Default Simulation Parameters for Cognitive Cloud Federation System

cloud service provider cs	10
code quantity pc	8
data set quantity dc	8
code-data relationship probability p	0.1
computational requirement C (GB)	$1 \sim 30$
size of data sets F_d (GB)	$1024 \sim 10240$
size of message exchange M (GB)	$0 \sim 0.1$
data access proportion ω_d	1×10^{-5}
minimum computing pricing P_α	$0.0073 \sim 0.0089$
maximum computing pricing P_α	$0.036 \sim 0.044$
minimum storage pricing P_β	$0.000032 \sim 0.000040$
maximum storage pricing P_β	$0.000050 \sim 0.000062$
minimum inbound network pricing P_ρ	$0.0011 \sim 0.0013$
maximum inbound network pricing P_ρ	$0.0019 \sim 0.0023$
minimum outbound network pricing P_θ	$0.11 \sim 0.13$
maximum outbound network pricing P_θ	$0.19 \sim 0.23$
code and data set privacy level (Θ, Φ)	$(4,4)$

This section describes the initial settings for our simulations. For user applications to be deployed over our cognitive cloud federation system, we specify the quantities of codes and data sets, while initiate their code-data relationship by randomly generating non-zero values for all elements in the matrix L with a probability of p. For the prices from different cloud service providers, we set up random values for minimum and maximum prices for a specific service and simulate the dynamic prices within the intervals. Note that the prices of computational resource is given by dollars per unit per hour, the prices of storage space is given by dollars per GB per hour and the prices of network is given by dollars per GB. From existing commercial cloud service pricing policies, the initial values for parameters of the simulations are set in Table 4.1. Note that, all random parameters follow uniform distributions.

4.3.2 Pricing Optimization

We first simulate the starting stage of the system to demonstrate the efficiency of seeking an optimal assignment with the lowest cloud service fee. In addition to our proposed optimization approach, we also compare the cloud service cost with our cloud federation system with the conventional *Single-Cloud* solution, which selects the cloud with lowest total service fee to host all codes and data sets.

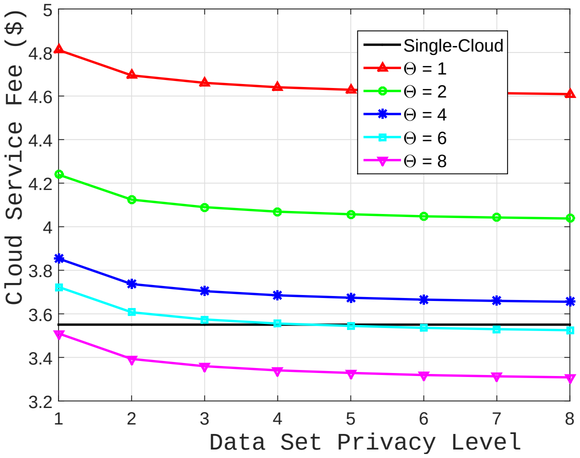

Figure 4.3: Tradeoff between Security Level and Cloud Service Fee

Figure 4.3 reveals the trade-off between privacy regulation and pricing optimization. We derive the cloud service fee for various combinations of privacy levels in codes and data sets. It is obvious that as the privacy level increases, either in codes Θ or in data sets Φ, the total cloud service fee grows. This requires the user to choose from different privacy levels based on various requirements. According to our experimental settings, the proposed cognitive optimization can only outperform the *Single-Cloud* system with $\Theta = 6$ and $\Phi > 6$.

One of the most important features of our cloud federation system is the data-code relationships. Figure 4.4 shows its impact on the cloud service fee. We increase the probability of code-data relationship, p, to illustrates its impact on cloud service fee. Note that, along with the growth of p from 0 to 1, the cloud service fees for distinct combinations of Θ and Φ all linearly raised to a higher level. Apparently, these increases are caused by additional inter-cloud data transmissions from data chunks

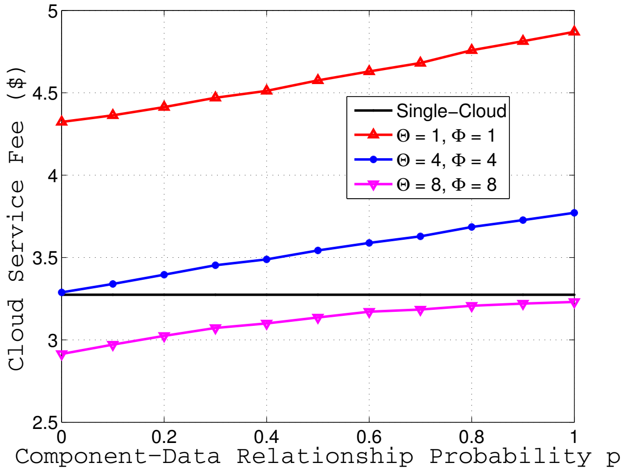

Figure 4.4: Effect of Data-Code Relationship on the Cloud Service Fee

to program codes. Similar comparison on Θ and Φ combinations are also conducted. As depicted, if we select the highest privacy level with $\Theta = 1$ and $\Phi = 1$, the cloud service fee is higher than that of *Single-Cloud* within the range of $30\% \sim 46\%$. In contrast, the setting of lowest privacy level with $\Theta = 8$ and $\Phi = 8$ will save the users' cost by $3\% \sim 20\%$.

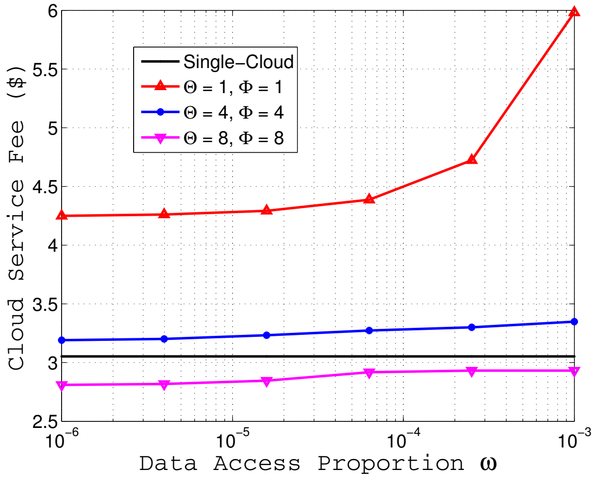

Figure 4.5: Effect of Data Access Proportion on the Cloud Service Fee

Another critical feature that impacts the cloud service cost is the data access proportion ω_d. As shown in Figure 4.5, we evaluate the system performance with various values of ω_d. Apparently, a larger ω_d yields a higher volume of data transmissions between multiple clouds, which results in a higher cloud service fee. Note that the cost for highest privacy level with $\Theta = 1$ and $\Phi = 1$ dramatically climbs to 6 dollars. The growth rate is relatively higher than other schemes. This is because the highest privacy level restricts the program codes and data chunks to be completely distributed, and thus the solution space for price-aware optimization is much smaller than others.

4.3.3 Cognitive Price-Aware Transition

After the establishment of the optimal assignment solution, we perform simulations over time to evaluate the platform's cognitive capability to the dynamic prices. With the initial privacy level setting at $\Theta = 4$ and $\Phi = 4$, we compare the cost of three deployment methodologies: *Single-Cloud:* to host all program codes and data chunks in the lowest-cost cloud and never change deployment over time, *Optimal-Static:* to find the optimal program codes and data chunks assignments initially and never change deployment over time, and *Optimal-Cognitive:* to be cognitive to the dynamic prices of multiple clouds and keep optimizing deployment assignment over time.

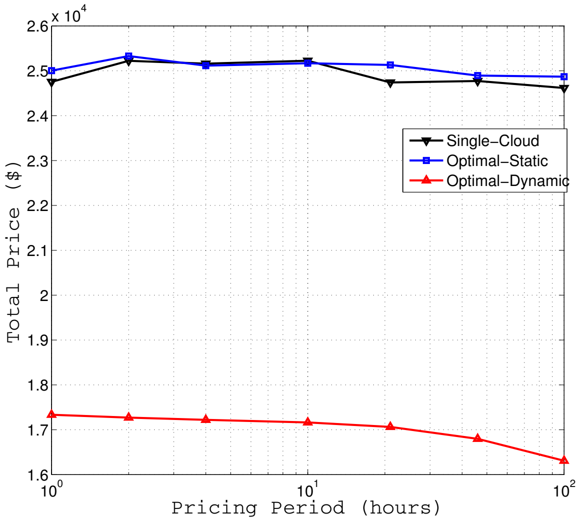

Figure 4.6: Efficiency of Proposed Platform with Pricing Variety Time Interval

According to our discussion before, the length of price change time interval will impact the state transition decisions. In our formulation, the optimization target in state transition involves the price change time interval t, which implies that longer time interval will make the cost in state transition worthy. Therefore, we expect to see a decrease in total cost with a longer interval in our simulations. Figure 4.6 illustrates the results of our experimental settings with value of t ranging from 1 hour to 100 hours. In contrary to the total cost of Single-Cloud and Optimal-Static methodologies, we observe a decline in the service cost of the Optimal-Cognitive scheme. In a nutshell, the performance of proposed dynamic optimization is not severely impacted by the frequency of price fluctuation. This phenomenon indicates that the occurrence of state-transition in our simulations rarely involves inter-cloud transfer of data chunks, which will instead significantly increase the cost of state-transition.

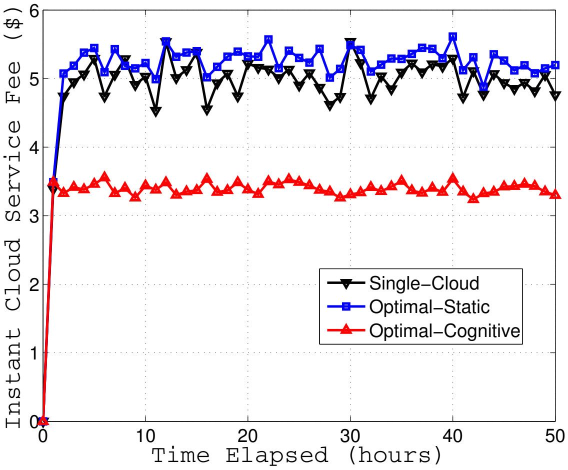

Figure 4.7: Efficiency of Proposed Platform with Time Elapsed

Given price change time interval $t = 1$, Figure 4.7 illustrates the average cloud service cost over 50 hours for 100 random iterations. In the first hour, *Optimal-Static* and *Optimal-Cognitive* share the same assignment for program codes and data chunks. So they share identical costs. Afterwards, since the *Single-Cloud* and *Optimal-Static* schemes are not cognitive to the price fluctuations of clouds, their performances become worse as time progresses. In contrast, the *Optimal-Cognitive* scheme dynamically adapts new strategies for different pricing combination, thus yields an around 30% reduction on total cloud service cost.

4.4 Summary

In this chapter, to make our dynamic pricing methodology more user friendly, we have investigated a price-aware cloud federation system that automatically chooses and cognitively migrates user tasks to a cloud provider charging at a more affordable rate as a reaction to dynamic prices. With the ability of migrating to a lower-rated cloud service, our cloud federation system provides a lower priced solution that benefits its customers financially. We have mathematically formulated and derived optimal solutions in both first-stage pricing minimization and cognitive strategy. Preliminary numeric results have revealed the trade-off between privacy and cost. Simulation results have also demonstrated the efficiency of the proposed system in reducing total cloud service cost while considering privacy constraints.

Chapter 5

Conclusion

5.1 Summary of Contributions

This thesis has tackled resource management of the cloud system through demand control using dynamic prices. Low server utilization problem cloud providers encountered today has numerous causes, including uneven application fit where the application cannot fully utilize allocated resources and the uncertainty in demand forecasts that requires the cloud system always need to be prepared for the demand burst during peak period. Many research works have been devoted to optimize resource allocation from different perspectives. In this thesis, we have introduced a new approach of using dynamic pricing to control aggregate demand to avoid resource over/under provisioning and achieve a higher profitability. Then, to free users from paying close attentions to the frequent changing prices and make real-time request decisions themselves, we have introduced a framework of a price-aware cloud federation system that automatically chooses and cognitively reacts to the dynamic prices and migrate user tasks that is charging at an affordable rate.

In Chapter 3 we have studied our pricing methodology with resource allocation techniques Knapsack algorithm and DRFH fair sharing algorithm. We have verified the correctness of our pricing methodology and simulation results have shown that the proposed pricing methodology is able to control demand as the resource allocation algorithm suggests.

In Chapter 4 we have formulated the price-aware cognitive decision making problem mathematically for our cloud federation system. We have demonstrated the feasibility and efficiency of the cloud federation system with in-depth simulations.

5.2 Future Directions

For potential extensions of the current work, we suggest to let users define the maximum prices they would accept for using a cloud service. Then, instead of responding to every price changes, the cloud federation system only need to reassess the cost of new code/data set assignment when the price exceeds the maximum prices. Also, we suggest to explore a more sophisticated system that allows redundant presence of data chunks and parallel codes over multiple clouds. With the duplicated copies, the cloud federation system will be more robust and reliable. Given the new setting, the pricing optimization procedure will a become more complicated system since the system ought to select the appropriate piece of program codes or data chunks from redundant identical copies. Note that, the information redundancy provides potential reduction in networking costs as it decreases the opportunity of intra-cloud transmission between program codes and data chunks. On the other hand, this redundancy introduces storage overheads to the system, which also raises the cost of storage spaces. Therefore, a well-designed mechanism that leverages this trade-off is needed.

Bibliography

[1] A. Greenberg, J. Hamilton, D.A. Maltz, and P. Patel, "The cost of a cloud: Research problems in data center networks," *SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 1, pp. 68–73, Dec. 2008.

[2] C. Weinhardt, A. Anandasivam, B. Blau, N. Borissov, T. Meinl, W. Michalk, and J. Ster, "Cloud computing-a classification, business models, and research directions," *Business & Information Systems Engineering*, vol. 1, no. 5, pp. 391–399, 2009.

[3] T. Nagle, J. Hogan, and J. Zale, *The Strategy and Tactics of Pricing*, vol. 1, Prentic Hall, United States, fifth edition, 2010.

[4] N. Bobroff, A. Kochut, and K. Beaty, "Dynamic placement of virtual machines for managing sla violations," in *Integrated Network Management, 2007. IM '07. 10th IFIP/IEEE International Symposium on*, May 2007, pp. 119–128.

[5] X. Meng, V. Pappas, and L. Zhang, "Improving the scalability of data center networks with traffic-aware virtual machine placement," in *INFOCOM, 2010 Proceedings IEEE*, March 2010, pp. 1–9.

[6] M.A. Sharkh, M. Jammal, A. Shami, and A. Ouda, "Resource allocation in a network-based cloud computing environment: design challenges," *Communications Magazine, IEEE*, vol. 51, no. 11, pp. 46–52, November 2013.

[7] H. Zhang, C. Jiang, N.C. Beaulieu, X. Chu, X. Wang, and T.Q.S. Quek, "Resource allocation for cognitive small cell networks: A cooperative bargaining game theoretic approach," *Wireless Communications, IEEE Transactions on*, vol. 14, no. 6, pp. 3481–3493, June 2015.

[8] D. Bernstein, D. Vij, and S. Diamond, "An intercloud cloud computing economy - technology,

governance, and market blueprints," in *SRII Global Conference (SRII), 2011 Annual*, March 2011, pp. 293–299.

[9] C. Hyser, B. Mckee, R. Gardner, and B.J. Watson, "Autonomic virtual machine placement in the data center," *Hewlett Packard Laboratories, Tech. Rep. HPL-2007-189*, pp. 2007–189, 2007.

[10] T. Dillon, Chen Wu, and E. Chang, "Cloud computing: Issues and challenges," in *Advanced Information Networking and Applications (AINA), 2010 24th IEEE International Conference on*, April 2010, pp. 27–33.

[11] S.K. Mandal, *On-Demand VM Placement on Cloud Infrastructure*, Ph.D. thesis, 2013.

[12] H.N. Van, F.D. Tran, and J.M. Menaud, "Sla-aware virtual resource management for cloud infrastructures," in *Computer and Information Technology, 2009. CIT '09. Ninth IEEE International Conference on*, Oct 2009, vol. 1, pp. 357–362.

[13] V. Mann, A. Kumar, P. Dutta, and S. Kalyanaraman, "Vmflow: Leveraging vm mobility to reduce network power costs in data centers," in *NETWORKING 2011*, J. Domingo-Pascual, P. Manzoni, S. Palazzo, A. Pont, and C. Scoglio, Eds., vol. 6640 of *Lecture Notes in Computer Science*, pp. 198–211. Springer Berlin Heidelberg, 2011.

[14] O. Biran, A. Corradi, M. Fanelli, L. Foschini, A. Nus, D. Raz, and E. Silvera, "A stable network-aware vm placement for cloud systems," in *Cluster, Cloud and Grid Computing (CCGrid), 2012 12th IEEE/ACM International Symposium on*, May 2012, pp. 498–506.

[15] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, and I. Stoica, "Dominant resource fairness: Fair allocation of multiple resource types.," in *NSDI*, 2011, vol. 11, pp. 24–24.

[16] W. Wang, B. Li, and B. Liang, "Dominant resource fairness in cloud computing systems with heterogeneous servers," in *INFOCOM*, 2014.

[17] T. Tomita and S. Kuribayashi, "Congestion control method with fair resource allocation for cloud computing environments," in *Communications, Computers and Signal Processing (PacRim), 2011 IEEE Pacific Rim Conference on*, Aug 2011, pp. 1–6.

[18] C. Joe-Wong, S. Sen, T. Lan, and M. Chiang, "Multi-resource allocation: Fairness-efficiency tradeoffs in a unifying framework," in *INFOCOM, 2012 Proceedings IEEE*, March 2012, pp. 1206–1214.

[19] M. Al-Roomi, S. Al-Ebrahim, S. Buqrais, and I. Ahmad, "Cloud computing pricing models: A survey.," *International Journal of Grid & Distributed Computing*, vol. 6, no. 5, 2013.

[20] J. Wang, M. Chen, and V.C.M. Leung, "A price-based approach to optimize resource sharing between cellular data networks and wlans," *Telecommunication Systems*, vol. 52, no. 2, pp. 485–496, 2013.

[21] S. Ha, S. Sen, C. Joe-Wong, Y. Im, and M. Chiang, "Tube: Time-dependent pricing for mobile data," *SIGCOMM Comput. Commun. Rev.*, vol. 42, no. 4, pp. 247–258, Aug. 2012.

[22] A. Narayan, S. Rao, G. Ranjan, and K. Dheenadayalan, "Smart metering of cloud services," in *Systems Conference (SysCon), 2012 IEEE International*, March 2012, pp. 1–7.

[23] S. Shang, J. Jiang, Y. Wu, Z. Huang, G. Yang, and W. Zheng, "Dabgpm: A double auction bayesian game-based pricing model in cloud market," in *Network and Parallel Computing*, C. Ding, Z. Shao, and R. Zheng, Eds., vol. 6289 of *Lecture Notes in Computer Science*, pp. 155–164. Springer Berlin Heidelberg, 2010.

[24] H. Shen and Z. Li, "New bandwidth sharing and pricing policies to achieve a win-win situation for cloud provider and tenants," in *INFOCOM, 2014 Proceedings IEEE*, April 2014, pp. 835–843.

[25] X. Jin, Y.K. Kwok, and Y. Yan, "A study of competitive cloud resource pricing under a smart grid environment," in *Proceedings of the 2013 IEEE International Conference on Cloud Computing Technology and Science - Volume 01*, Washington, DC, USA, 2013, CLOUDCOM '13, pp. 655–662, IEEE Computer Society.

[26] A. Gohad, N.C. Narendra, and P. Ramachandran, "Cloud pricing models: A survey and position paper.," in *Cloud Computing in Emerging Markets (CCEM), 2013 IEEE International Conference on*, Oct 2013, pp. 1–8.

[27] B. Sharma, R.K. Thulasiram, P. Thulasiraman, S.K. Garg, and R. Buyya, "Pricing cloud compute commodities: A novel financial economic model," in *Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (Ccgrid 2012)*, Washington, DC, USA, 2012, CCGRID '12, pp. 451–457, IEEE Computer Society.

[28] G. Gallego and G. Van Ryzin, "Optimal dynamic pricing of inventories with stochastic demand over finite horizons," *Management Science*, vol. 40, no. 8, pp. 999–1020, 1994.

[29] W. Elmaghraby and P. Keskinocak, "Dynamic pricing in the presence of inventory considerations: Research overview, current practices, and future directions," *Management Science*, vol. 49, no. 10, pp. 1287–1309, 2003.

[30] O.A. Ben-Yehuda, M. Ben-Yehuda, A. Schuster, and D. Tsafrir, "Deconstructing amazon ec2 spot instance pricing," *ACM Transactions on Economics and Computation*, vol. 1, no. 3, pp. 16:1–16:20, Sept. 2013.

[31] W. Lu, S. Chen, K. Li, and L.V. S. Lakshmanan, "Show me the money: Dynamic recommendations for revenue maximization," in *the Very Large Data Bases Endowment*, Sep 2014, vol. 7.

[32] M. Zhang, R. Ranjan, M. Menzel, S. Nepal, P. Strazdins, W. Jie, and L. Wang, "An infrastructure service recommendation system for cloud applications with real-timeqosrequirement constraints," *Systems Journal, IEEE*, 2015.

[33] A. Alfazi, T.H. Noor, Q.Z. Sheng, and Y. Xu, "Towards ontology-enhanced cloud services discovery," in *Advanced Data Mining and Applications*, X. Luo, J.X. Yu, and Z. Li, Eds., vol. 8933 of *Lecture Notes in Computer Science*, pp. 616–629. Springer International Publishing, 2014.

[34] D. Bernstein, E. Ludvigson, K. Sankar, S. Diamond, and M. Morrow, "Blueprint for the intercloud - protocols and formats for cloud computing interoperability," in *Internet and Web Applications and Services, 2009. ICIW '09. Fourth International Conference on*, May 2009, pp. 328–336.

[35] R. Ranjan and R. Buyya, "Decentralized overlay for federation of enterprise clouds," *The Handbook of Research on Scalable Computing Technologies*, pp. 191 – 217, 2009.

[36] N. Grozev and R. Buyya, "Inter-cloud architectures and application brokering: taxonomy and survey," *Software: Practice and Experience*, vol. 44, no. 3, pp. 369–390, 2014.

[37] A. Toosi, R. Calheiros, R. Thulasiram, and R. Buyya, "Resource provisioning policies to increase iaas provider's profit in a federated cloud environment," in *High Performance Computing and Communications (HPCC), 2011 IEEE 13th International Conference on*, Sept 2011, pp. 279–287.

[38] R. Buyya, R. Ranjan, and R.N. Calheiros, "Intercloud: Utility-oriented federation of cloud computing environments for scaling of application services," in *Proceedings of the 10th International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP 2010)*. pp. 21–23, Springer.

[39] J. Ekanayake and G. Fox, "High performance parallel computing with clouds and cloud technologies," vol. 34 of *Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering*, pp. 20–38. Springer Berlin Heidelberg, 2010.

[40] T.A. Johnson, R. Eigenmann, and T. N. Vijaykumar, "Min-cut program decomposition for thread-level speculation," in *Proceedings of the ACM SIGPLAN 2004 Conference on Programming Language Design and Implementation*, New York, NY, USA, 2004, PLDI '04, pp. 59–70, ACM.

[41] D.W. Watson, J.K. Antonio, H.J. Siegel, and M.J. Atallah, "Static program decomposition among machines in an simd/spmd heterogeneous environment with non-constant mode switching costs," in *Heterogeneous Computing Workshop, 1994., Proceedings*, Apr 1994, pp. 58–65.

[42] I. Giurgiu, O. Riva, D. Juric, I. Krivulev, and G. Alonso, "Calling the cloud: enabling mobile phones as interfaces to cloud applications," in *Proceedings of the ACM/IFIP/USENIX 10th international conference on Middleware*, Berlin, Heidelberg, 2009, Middleware'09, pp. 83–102.

[43] B. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "Clonecloud: elastic execution between mobile device and cloud," in *Proceedings of the sixth conference on Computer systems*, New York, NY, USA, 2011, EuroSys '11, pp. 301–314.

[44] W. Cai, C. Zhou, V. Leung, and M. Chen, "A cognitive platform for mobile cloud gaming," in *2013 IEEE 5th International Conference on Cloud Computing Technology and Science (CloudCom 2013)*, 2013.

[45] K. Zhang, X. Zhou, Y. Chen, X. Wang, and Y. Ruan, "Sedic: privacy-aware data intensive computing on hybrid clouds," in *Proceedings of the 18th ACM conference on Computer and communications security*. ACM, 2011, pp. 515–526.

[46] L. Lamport, R. Shostak, and M. Pease, "The byzantine generals problem," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 4, no. 3, pp. 382–401, 1982.

[47] A. Bessani, M. Correia, B. Quaresma, F. André, and P. Sousa, "Depsky: dependable and secure storage in a cloud-of-clouds," *ACM Transactions on Storage (TOS)*, vol. 9, no. 4, pp. 12, 2013.

[48] M. Schnjakin, D. Korsch, M. Schoenberg, and C. Meinel, "Implementation of a secure and reliable storage above the untrusted clouds," in *Computer Science Education (ICCSE), 2013 8th International Conference on*, April 2013, pp. 347–353.

[49] W. Ma, H. Zhang, Zheng.W., and X. Wen, "Differentiated-pricing based power allocation in dense femtocell networks," in *Wireless Personal Multimedia Communications (WPMC), 2012 15th International Symposium on*, Sept 2012, pp. 599–603.

[50] H. Zhang, C. Jiang, X. Mao, and H. Chen, "Interference-limited resource optimization in cognitive femtocells with fairness and imperfect spectrum sensing," *Vehicular Technology, IEEE Transactions on*, vol. PP, no. 99, pp. 1–1, 2015.

[51] R. Harmon, D. Raffo, and S. Faulk, "Incorporating price sensitivity measurement into the software engineering process," in *Management of Engineering and Technology, 2003. PICMET '03. Technology Management for Reshaping the World. Portland International Conference on*, July 2003, pp. 316–323.

[52] R.E. Goldsmith and S.J. Newell, "Innovativeness and price sensitivity: managerial, theoretical and

methodological issues," *Journal of Product & Brand Management*, vol. 6, no. 3, pp. 163–174, 1997.

[53] S. Han, S. Gupta, and D.R. Lehmann, "Consumer price sensitivity and price thresholds," *Journal of Retailing*, vol. 77, no. 4, pp. 435 – 456, 2001.

[54] D. Gale, "The law of supply and demand," *MATHEMATICA SCANDINAVICA*, vol. 3, pp. 155–169, 1955.

[55] H. Xu and B. Li, "A study of pricing for cloud resources," *SIGMETRICS Perform. Eval. Rev.*, vol. 40, no. 4, pp. 3–12, Apr. 2013.

[56] P. Jacko, "Resource capacity allocation to stochastic dynamic competitors: knapsack problem for perishable items and index-knapsack heuristic," *Annals of Operations Research*, pp. 1–25, 2013.

[57] K. Pan, W. Liu, and H. Li, "Work in progress: On the knapsack-based resource allocation for reconfigurable network architecture," in *Communications and Networking in China (CHINACOM), 2014 9th International Conference on*, Aug 2014, pp. 621–624.

[58] H.R. Varian, "Equity, envy, and efficiency," *Journal of Economic Theory*, vol. 9, no. 1, pp. 63 – 91, 1974.

[59] H. Wang, Q. Jing, R. Chen, B. He, Z. Qian, and L. Zhou, "Distributed systems meet economics: pricing in the cloud," in *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*. USENIX Association, 2010, pp. 6–6.

[60] M. Armbrust, A. Fox, R. Griffith, A.D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of cloud computing," *Commun. ACM*, vol. 53, no. 4, pp. 50–58, Apr. 2010.

[61] M. Gall, A Schneider, and N. Fallenbeck, "An architecture for community clouds using concepts of the intercloud," in *Advanced Information Networking and Applications (AINA), 2013 IEEE 27th International Conference on*, March 2013, pp. 74–81.

[62] M. Schnjakin and C. Meinel, “Evaluation of cloud-raid: A secure and reliable storage above the clouds,” in *Computer Communications and Networks (ICCCN), 2013 22nd International Conference on*, July 2013, pp. 1–9.

[63] D. Lange and O. Mitsuru, *Programming and Deploying Java Mobile Agents Aglets*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1998.

[64] M. Mechtri, D. Zeghlache, E. Zekri, and I.J. Marshall, “Inter-cloud networking gateway architecture,” in *Cloud Computing Technology and Science (CloudCom), 2013 IEEE 5th International Conference on*, Dec 2013, vol. 2, pp. 188–194.