Broker-Mediated Multiple-Cloud Orchestration Mechanisms for Cloud Computing

Ganesh Neelakanta Iyer

Department of Electrical and Computer Engineering

National University of Singapore

A thesis submitted for the degree of

Doctor of Philosophy

2012

To my loving parents ...

Neelakanta Iyer
Vasantha

Acknowledgements

I wish to express my deep and sincere appreciation to my supervisor, Professor Bharadwaj Veeravalli, for his guidance, help and support. It is Professor Bharadwaj who planted the seed for exciting research in Cloud Computing. I would like to gratefully and sincerely thank him for his guidance, understanding, patience, and most importantly, his friendship during my graduate studies at NUS. His mentorship was paramount in providing a well rounded experience consistent my long-term career goals. He encouraged me to not only grow as an applied researcher but also as an instructor and an independent thinker. I would probably have been lost without him and his style of guidance.

I would like to thank Dr Peng-Yong Kong who introduced me to the interesting world of game theory and economic models for computer engineering. I would like to thank members of my thesis committee Prof Cheong Loong Fah and Dr Marc Armand for their encouragement, insightful comments, and hard questions.

Special thanks to my friends Mingding, Yuncai, Dr Lingfang, Sakthiganesh, Raghavendran, Dinesh, Sivakumar, Vaishali, Li Xiao, Ramkumar, Maitreya, Srikanth, Balaji and Anupkumar for several useful discussions and also helping me in my research in different ways.

My time at NUS was made enjoyable in large part due to the many friends and groups that became a part of my life. I am grateful for time spent with roommates and friends, for my travel buddies and our memorable trips to different countries in south east Asia and for many other people and memories. Special thanks to my friends Mridul, Chaitanya, Jerrin, Deepu, Manmohan, Abhilash and Pramod for several useful discussions over lunch and tea at Dilys.

I would also like to thank all my teachers in Bhaskars Academy who made me continue my passion for Kathakali and other traditions while carrying out my research. I specially thank my Kathakali Guru Kalamandalam Biju and his wife Mayadevi for making me not missing my home. My special thanks to my teachers Bhaskar Uncle, Santha Bhaskar aunt, Sajith Sir, Binsin Teacher and Harikrishnan Sir.

Further I would like to thank my mentors and friends in Facilitators@NUS and ECE Graduate Student Council which helped myself to develop my personal skills. My special thanks to Mr Terence, Prof Leng Siew, Jaslin, Xiaolei and Yongfu.

Lastly, I would like to thank my family for all their love and encouragement. For my parents Neelakanta Iyer and Vasantha who raised me with a love of science and supported me in all my pursuits. I thank my wonderful brother Girish who is the best friend in my life. I thank my in-laws Narayana Swamy, Meenakshy, Revathi and Harikrishnan and other family members for all the support and encouragement throughout my studies. And most of all for my loving, supportive, encouraging, and patient wife Lakshmy for faithful support during the later stages of this Ph.D.

Contents

Summary

With a plethora of Cloud Service Providers (CSPs) offering various kinds of services, it is difficult for a user to choose an appropriate CSP or a set of CSPs for executing its tasks. Users are also concerned about other parameters such as security and trustworthiness of the CSPs. Further some of the user applications have tight requirements such as deadline and budget specifications and they need to be deployed among multiple CSPs to meet such requirements. On the other hand, CSPs currently follow fixed price per resource and they need efficient mechanisms to monitor the market and to develop attractive dynamic pricing strategies based on several parameters including user demand, competition and user profile.

In the first part of this thesis, we describe a comprehensive Cloud Broker architecture and focus on designing Broker-mediated Multiple-Cloud Orchestration mechanisms to connect various CSPs and users together. We propose three Broker-based Cloud service arbitrage mechanisms (Incentive based, Sealed-bid continuous double auction based and Risk based) for different types of applications in which the Broker supplies flexibility and opportunistic choices for users

and foster the competition between Clouds. Users can consider various parameters such as trust, reputation and security to choose an appropriate CSP. We also propose market-oriented dynamic pricing strategies for CSPs to adapt to market conditions quickly.

In the second part of this thesis, we propose two Cloud Broker aggregation mechanisms for IaaS Clouds where one is based on cooperative bargaining games and the other one is based on Markovian queues. In the first case, we employ bargaining solutions propounded in literature to efficiently determine the resource requirements for a set of tasks, requesting for one type of resources, so as to maximize the resource utilization and to handle elastic user requirements. It also introduces an asymmetric pricing mechanism to consider user's budget requirements. The Markovian queue based approach efficiently aggregates user tasks/data among Clouds with heterogeneous resource capabilities based on user's deadline and budget specifications. We further address the task scheduling within a Cloud to reduce the makespan and to improve the resource usage after the aggregation is completed. Our Broker can function either as an entity to connect several CSPs and users or as an entity to connect several users to one CSP and incorporates several features suitable for various situations and different types of users.

List of Figures

List of Tables

Acronyms

AWS	Amazon Web Services
B2B	Business to Business
BoT	Bag-of-Tasks
CDA	Continuous Double Auction
CPU	Central Processing Unit
CSP	Cloud Service Provider
DC	Datacenter
DLT	Divisible Load Theory
EC2	Elastic Compute Cloud
ERP	Enterprize Resource Planning
FCFS	First Come First Serve
IaaS	Infrastructure as a Service
JDM	Job Distribution Manager
LAN	Local Area Network
NBS	Nash Bargaining Solution
OM	Operations Monitor
OST	Optimal Sequence Theorem
PaaS	Platform as a Service

PM	Price Manager
PVR	Point Value Representation
QoS	Quality of Service
RBS	Raiffa - Kalai-Smorodinsky Bargaining Solution
RUI	Resource Usage Index
S3	Simple Storage Service
SaaS	Software as a Service
SAN	Storage Area Network
SLA	Service Level Agreement
SME	Small and Medium scale Enterprizes
SSO	Single Sign-On
VC	Virtual Cluster
VCI	Virtual CPU Instance
VIA	Virtual Interface Architecture
VM	Virtual Machine
VNM-UT	Von Neumann Morgenstern Utility Theory
VO	Virtual Organization
WAN	Wide Area Network

Notations

Table 2: Table of Major Notations

Notation	Meaning
PART 1: Cloud Broker Arbitrage Mechanisms	
N	Total number of users in the system
C_i	i^{th} user
M	Total number of CSPs in the system
P_j	j^{th} provider
RR_{ij}	Return Ratio of C_i maintained by P_j
ER_{ij}	Expected Return Ratio of C_i maintained by P_j
e_{ij}	Number of jobs from C_i executed by P_j
S_{ij}	Number of job requests submitted to P_j by C_i
Pr_{ij}^t	Price per resource of P_j for C_i at time t
X_{ij}	Offer price quoted by P_j for C_i
Y_{ij}	Affinity index of P_j by C_i
Z_{ij}	Security index of P_j by C_i
U_{ij}	Utility for C_i if P_j is chosen
E_i	Total expenditure for C_i
I_j	Gross income for P_j
D_i	Number of resources requested by user C_i
L_j^t	Load on P_j at time t
χ_i	Reliability factor of user C_i
H_j	Weighted average system security for P_j
R_{ij}	Reputation index of P_j by C_i

Continued on next page

Table 2 – continued from previous page

Notation	Meaning
T_{ij}	Trust index of P_j by C_i
φ	Threshold vale for load in CSP
χ_i^t	Reliability index for User C_i
$TR_{i,j}^r$	Job Rating of user i to CSP j for all past transactions with reference
$TR_{i,j}$	Job Rating of user i to CSP j for all past transactions without reference
$JR_{i,j}^n$	Job Rating of user i to CSP j for transaction n
$RC_{i,\gamma}^n$	Credit Rating of user i to reference user γ for reference transaction n
$RC_{i,\gamma}$	Credit Rating of user i to reference user γ for all past reference transactions
V	Utility function based on Trust T
U	Utility function for the user based on trust and cost
E_c	Explicit cost involved in the current transaction
δ	Discount factor for calculating utility based price offers
E_{cd}	Explicit cost for the current transaction with discounting
P_t	Price offered by a CSP at time t
Q_t	Average price other CSPs offered at time t
S_t	Amount of resources sold at time t
O_t	Amount of resources offered at time t
A_{thr}	Expected acceptance rate for CSP
A_t	Actual acceptance rate for CSP
ξ	the weighing parameter for past price and reference price
W	User's budget
PART 2 Cloud Broker Aggregation Mechanisms	
N	Total number of tasks
N_j	Number of tasks in CSP j
B	Budget specified by the user
R_i^*	Optimal number of resources derived based on NBS
$\tilde{R}_i^*$	Optimal number of resources derived based on RBS
d	Disagreement Point
α_i	Bargaining power for player i
R_{tot}	Total number of resources available for allocation
μ_0	Rate of dispatcher
p_j	Fraction of job sent to CSP j

Continued on next page

Table 2 – continued from previous page

Notation	Meaning
λ_j	Arrival rate of tasks in CSP j
μ_j	Service Rate of CSP j
t_j	Average time a task spent in CSP j
T_j	Total amount of time spent by all tasks that are assigned to CSP j
D_j	Unit cost per resource per time unit
D_{tot}	Total expenditure for the execution of one BoT application
A	Total number of BoT applications
c_k	Monetary cost/computation time unit of VM_k expressed in dollars
K	Number of virtual clusters (VCs)
e_k	Expected number of tasks
l_0	Expected number of *certain* tasks
l_1	Expected number of *uncertain* tasks
ρ_0	System utilization for *certain* tasks in a VC
ρ_1	System utilization for *uncertain* tasks in a VC
n_k	Number of VMs in VC_k
m_k	Maximum number of VMs that can be allocated to *uncertain* tasks in VC_k
T_k^0	Execution time for all *certain* tasks in VC_k
T_k^1	Execution time for all *uncertain* tasks in VC_k
M_c	Makespan of *certain* tasks
ψ	Resource Usage Index
Appendix	
m+1	Total Number of processors including the resource allocator VCI
p_i	i *th* processor
T_{cm}	Time taken to transmit a unit load by the communication link
T_{cp}	Time taken to process a unit load by a VCI
α_i	Fraction of the load assigned to VCI p_i.
w_i	The inverse of the computation speed of VCI p_i
z_i	The inverse of the communication speed of the link i
E_i	The product $w_i T_{cp}$ referring to the time taken to process a unit load by p_i
C_i	The product $z_i T_{cm}$ referring to the time taken to transfer a unit load via link l_i

Continued on next page

Table 2 – continued from previous page

Notation	Meaning
θ_{cm}	An additive communication overhead component that includes the sum of all delays associated with the communication process.
θ_{cp}	An additive computation overhead component that includes the sum of all delays associated with the computation process
A_i^l	Availability of link i
A_i^p	Availability of VCI i
L	Load that will be distributed to all VCIs for processing

Chapter 1

Introduction

Cloud Computing has been emerged as an attractive paradigm for small, medium and large scale business enterprises due to its inherent characteristics. Cloud Computing can be defined as a model which delivers applications as services (known as Software as a service or SaaS) over internet and providing hardware and system software for users to implement, deploy and maintain their custom-made applications and/or services [6].

There are five essential characteristics for Cloud environments [7]. Firstly, Resources can be provisioned *rapidly on-demand* and customers can configure the Cloud resources as needed automatically. Secondly, Cloud allows a *broad network access* wherein users can access and use Cloud resources through network using various heterogeneous client devices such as mobile phones and laptops. Thirdly, Cloud Service Providers (CSPs) use *Virtualization* techniques to pool the computing resources to serve multiple consumers based on user demand. Fourthly, the *auto-elasticity* of Cloud allows users to configure resources in minutes and enables them to scale the capacity based on their instant resource requirements

elastically. Finally, Cloud Computing has an attractive utility computing based *pay-as-you-go* policy in which a user needs to pay only for the capacity that he/she actually uses.

Cloud Computing has several key differences with respect to other traditional distributed computing models such as Grid Computing and Cluster Computing. A computing Cluster [3] consists of interconnected stand-alone computers which works cooperatively as a single interconnected computing resource. In Grid Computing, resources from several locations are connected via high-speed network links and allows close interactions among the applications running. In case of Cloud Computing, workloads can be quickly scaled out through on-demand resource provisioning of virtual and/or physical resources which is characterized by several key features such as failure handling via VM migration, utility computing model and resource monitoring. A classification of key characteristics of these distributed computing systems are summarized in Table 1.1.

1.1 Cloud Service Delivery Models

Cloud Computing paradigm is characterized by three main service models as described below:

Software as a Service (SaaS) SaaS delivers software and data as a service over internet which are accessible from various client devices. Customers do not need to buy software licences or additional infrastructure equipment, but they need to pay for what they use. There are both free as well as paid applications delivered in this way. Examples include Google Apps [8] and

Table 1.1: Classification of Network-Based Computing Systems ([3], [4])

Functionality, Applications	**Cluster Computing**	**Grid Computing**	**Cloud Computing**
Architecture, Network Connectivity and Size	Network of computer Nodes interconnected by SAN, LAN or WAN hierarchically	Heterogeneous clusters interconnected by high-speed network links	Virtualized clusters of servers over data centers via SLA
Control and Resource Management	Homogeneous nodes with distributed control, running UNIX or LINUX	Centralized control, server-oriented with authenticated security	Dynamic resource provisioning of servers, storage and networks
Security and Privacy	Traditional login/password-based. Medium level of privacy	Public/private key pair based authentication and mapping a user to an account. Limited support for privacy.	Each user/application is provided with a virtual machine. High security/privacy is guaranteed.
Service Negotiation	Limited	Yes, SLA based	Yes, SLA based
User Management	Centralized	Decentralized and also Virtual Organization (VO)-based	Centralized or can be delegated to third party
Resource Management	Centralized	Distributed	Centralized/Distributed
Allocation / Scheduling	Centralized	Decentralized	Both centralized/ decentralized
Standards / Inter-Operability	Virtual Interface Architecture (VIA)-based	Some Open Grid Forum standards	Web Services (SOAP and REST)
Capacity	Stable and guaranteed	Varies, but high	Provisioned on demand
Failure Management (Self-healing)	Limited (often failed tasks/applications are restarted).	Limited (often failed tasks/applications are restarted).	Strong support for failover and content replication. VM migration is possible
Pricing of Services	Limited, not open market	Dominated by public good or privately assigned	Utility pricing, pay-as-you-go, dynamic strategies like spot pricing
Internetworking	Multi-clustering within an Organization	Limited adoption, but being explored through research efforts such as Gridbus InterGrid	High potential, third party providers can loosely tie together services of different Clouds
Potential for building third-party value-added solutions	Limited due to rigid architecture	Limited due to strong orientation for scientific computing	High potential can create new services by dynamic provisioning of different services and offer as their own Cloud services to users
Application and Network-centric services	High performance computing, search engines, web services etc.	Distributed super computing, global problem solving	utility computing, outsourced computing services and elastic applications
Representative operational systems	Google search engine, SunBlade, IBM road Runner	TeraGrid, UK EGEE, D-grid	Google App Engine, Amazon Web Services, IBM Blue-Cloud

sales management applications by Salesforce.com[9].

Platform as a Service (PaaS) PaaS delivers a development platform as well as a solution stack on demand. Users can rent virtualized servers and services to develop, test and run their applications. It allows developers from different parts of the world to work together on software development projects. With the PaaS model, the developer needs the knowledge only to integrate various building blocks of a project such as the hardware, operating system, database etc, leaving minor details to be taken care by the CSP. PaaS is also used to enhance the capabilities of the applications developed as SaaS. Some typical examples include Google App Engine [10] and Microsoft Azure Platform [11].

Infrastructure as a Service (IaaS) In IaaS, CSPs offer all the tools necessary for building, deploying and extending their custom-made applications and services. Offered equipment include storage hardware, servers and networking components owned and maintained by the CSPs. The user pays based on the actual usage of the resources. It serves as a foundation for PaaS and SaaS models which is flexible, standard, and virtualized operating environment. Clients have more options to customize their applications compared to PaaS and IaaS. A typical example is Amazon Elastic Compute Cloud (EC2) [12]. In Amazon EC2, you can develop and execute your applications on a virtual computer (also known as Virtual Instance) with a specific configuration. A typical standard large instance of Amazon is 7.5 GB of memory, 4 EC2 Compute Units (2 virtual cores with 2 EC2 Compute Units each), 850 GB of local instance storage and 64-bit platform.

1.2 Key Challenges in Cloud Computing

There are several key challenges for both users and providers to enter and establish in this new distributed computing paradigm. Key challenges faced by the users in moving their data/services to Cloud platforms include the following:

- **Choosing the right provider:** With the variety of services offered by several CSPs, users may find it difficult to choose the right provider which matches their requirements. At present, there is no platform which provides information about the capabilities of all the CSPs.

- **Security and Privacy issues:** As several users may share the same physical infrastructure in a virtualized manner simultaneously, users are often concerned about the security and privacy of their data in the Cloud platform. This is an important issue because, the data/service storage/running location specific information is abstracted from the users in Cloud environments.

- **Trustworthiness of CSPs:** Users are concerned about the trustworthiness of the CSPs. This aspect is different from security because, trustworthiness conveys information pertaining to the task execution such as adhering to Service-Level Agreements (SLA adherence) and reliability of task execution (such as handling node failure, meeting task deadline etc).

- **Dealing with lock-in:** In economics, vendor lock-in makes a customer dependent on a vendor for specific products and/or services making it difficult for users to choose another CSP without substantial switching costs. The switching cost includes possible end-of-contract penalties, charges for

format conversion and data/application switching and possible additional charges for bandwidth usage.

From the providers's perspective, there are many challenges to be addressed for exploiting various features of Cloud platforms. It include:

- **Understanding the market:** New Cloud providers may need to understand the current market status in terms of the competitors in the domain, the user preferences in terms of the products/services they prefer most of the time, user preferences for various features such as security and trust requirements etc.

- **Adapting to the market:** Current Cloud platforms follow a fixed price per resource for their products and services with some small exceptions like Amazon spot pricing [13]. Dynamic pricing strategies are required to improve their performance and to attract more customers based on the market situation.

- **Monitoring user profile:** With competition among different providers, CSPs may be required to monitor the reliability of users in terms of the feedback given by them to decide user acceptance criteria. It also helps to avoid any unhealthy competition among the providers and users.

1.3 Objectives and organization of the thesis

1.3.1 General focus, Contributions and Scope

This thesis focuses on developing a comprehensive architecture for a Cloud Broker and devising strategies for various Cloud Broker service models for multiple Cloud orchestration mechanisms. Our Broker architecture helps both users and CSPs to make their business decisions and addresses services for a variety of user applications. Our architecture also gives flexibility to add new services in the future based on future requirements.

We divide the thesis into two parts. First part proposes three strategies for Multiple-Cloud arbitrage mechanisms. These are based on incentives, sealed-bid double auction and Von Neumann-Morgenstern utility theory. All these strategies help users to choose an appropriate CSP based on their preferences. CSPs can make attractive dynamic price offers based on the market conditions. Through extensive performance evaluation, we study the effectiveness of these schemes under various cases and compare the performance with the current Cloud market without Cloud Broker for Cloud Service Arbitrage.

Second part of the thesis specifically focuses on Cloud Aggregation mechanisms for compute and data intensive IaaS applications such as BoT applications and large-scale divisible load applications. First, we propose two bargaining models based on cooperative game theoretic approaches propounded in literature for Cloud Aggregation which decide the task distribution for a set of tasks arriving at the Broker based on various parameters such as deadline and budget requirements. We then propose a Cloud aggregation based on Markovian queues which

can deploy user tasks into CSPs with heterogeneous resource capabilities to satisfy user requirements. Further, we address the task scheduling and its effect on a particular Cloud after the task aggregation is completed.

The scope of this thesis is to develop a comprehensive architecture model for a Cloud Broker and is to devise strategies for addressing various Cloud Broker service models to address different categories of users. We also propose demand-based dynamic pricing strategies for the CSPs to adapt to market situations quickly. We further show the effectiveness of our proposed models through extensive performance evaluation studies.

1.3.2 Outline of the thesis

In Chapter 2, we first describe the problem addressed by this work and the motivation. Then we provide a comprehensive survey of various various Cloud Broker mechanisms existing in the literature and classify them into three based on the services offered. Then the broker-based system architecture for multiple-Cloud orchestration is described in detail.

In Chapter 3, we propose two Cloud Service Arbitrage mechanisms that enable users to choose the right CSP and the CSPs to offer competitive price offers based on market conditions. One scheme is based on incentives whereas the other scheme is based on sealed-bid continuous double auctions. Incentive-based strategy is shown to be suitable for loyal users who use Cloud for their applications very often. The applications that fall under this include SaaS applications such as B2B and ERP applications and PaaS applications such as web hosting. The auction-based scheme is suitable for users who use Cloud on an ad-hoc basis

and for users with tighter budget requirements who can afford to wait for longer time to deploy and complete the execution of their applications.

In Chapter 4, we propose a Cloud Arbitrage mechanism based on risk and trust wherein users choose the right CSP based on various system parameters. This scheme is suitable for users who are more risk-averse in making their business decisions and higher flexibility is given to users to choose various parameters in calculating the Von Neumann-Morgenstern utility function. We also propose a dynamic pricing strategy based on acceptance rate and compare our scheme with the other proposed schemes as well as static pricing cases.

In Chapter 5, we propose a Cloud aggregation model for distributing tasks among compute resources (with similar characteristics) based on Cooperative game-theoretic approaches. We use two bargaining solutions propounded in the literature - Nash Bargaining Solution (NBS) and Raiffa Bargaining Solution (RBS) for Cloud aggregation and intermediation.

In Chapter 6, we propose a Broker mediated Cloud aggregation mechanism using Markovian Queues which can effectively deploy customer applications over multiple CSPs for Bag-of-Tasks (BoT) applications. We then analyze the task distribution and resource allocation within a datacenter using queueing theory and analyze the effectiveness of our model in various cases.

In Chapter 7, we conclude this thesis by providing a summary of all our works. We also describe possible/interesting future works based on this thesis.

Chapter 2

Problem Statement, Background and System Architecture

2.1 Problem Formulation and Motivation

We propose a comprehensive Cloud Broker architecture and strategies for Multiple-Cloud Orchestration based on the Broker architecture to solve several key issues faced by users and CSPs in Cloud Computing environments.

2.1.1 Need for Broker-based Cloud Orchestration mechanisms

As Cloud emerges as a competitive sourcing strategy, a demand is clearly arising for the integration of Cloud environments to create an end-to-end managed landscape of Cloud-based functions. A Broker-based multiple Cloud orchestration mechanism can solve most of the issues faced by both users as well as the CSPs.

Cloud orchestration relates to the connectivity of IT and business process levels between Cloud environments. Major benefits of Cloud Orchestration are:

- Helps users to choose the best service they are looking for
- Helps providers to offer better services and adapt to market conditions quickly
- Ability to create a best of breed service-based environment in which tasks can be dynamically deployed among multiple CSPs to reduce task execution time and to meet budget requirements
- Helps users and providers to make their business decisions based on several collective parameters such as trust, reputation, security and reliability which are difficult to handle in the absence of a Broker.
- Helps users to designate Broker to make some decisions on behalf of them so that users can focus on their core business rather than focusing on task deployment strategies and other system administration jobs.

2.1.2 Cloud Broker Service Models

Cloud Broker plays an intermediary role to help customers locate the best and the most cost-effective CSP for the customer needs. Cloud Broker is by far the best solution for Multiple Cloud Orchestration (includes aggregating, integrating, customizing and governing Cloud services) for SMEs and large enterprises. Major advantages are cost savings, information availability and market adaptation. As the number of CSPs continues to grow, a single interface (Broker) for information, combined with service, could be compelling to companies who prefer to spend

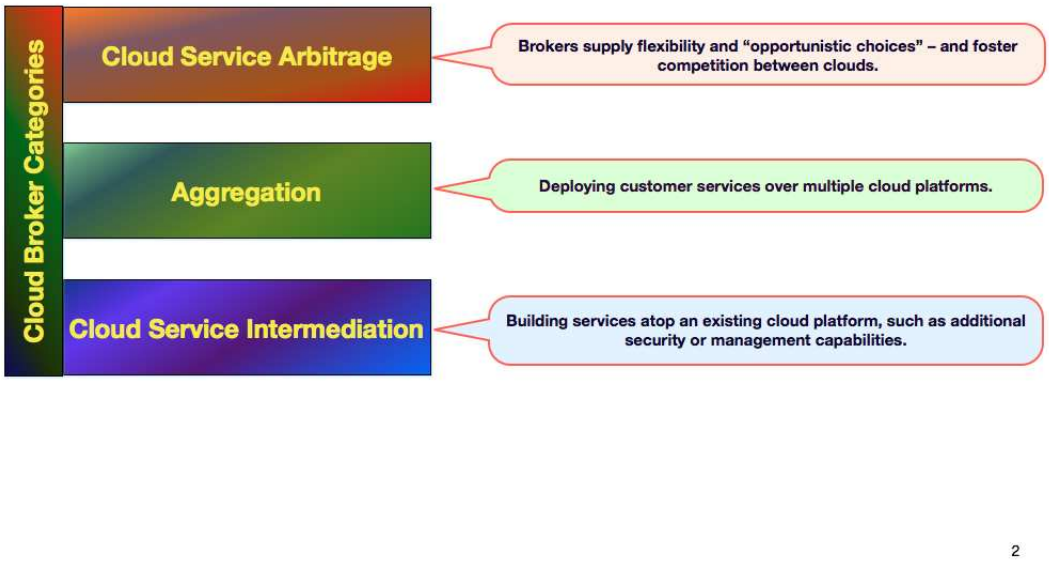


Figure 2.1: An overview of various Cloud Broker Mechanisms [1]

more time with their Clouds than devising their own strategies for finding the suitable CSP to meet their needs.

This is further corroborated by various research statistics. According to Gartner [14], "By 2015, at least 20 percent of all Cloud services will be handled via brokers, rather than directly, up from less than 5 percent today." Another research by Gartner states that, "Through 2014, Cloud service brokerage will generate more than $5 billion in sales, up from less than $50 million this year, making it the fastest growing area of Cloud Computing".

We can broadly classify Cloud Brokers into three based on the services offered

by them as illustrated in Figure 2.1. First one is *Cloud Service Intermediation*, wherein Broker can build services on top of the services offered by the CSPs, such as additional security features and/or management capabilities. Second one is *Aggregation* in which the Broker deploys customer services over multiple CSPs. Finally *Cloud Service Arbitrage* where the Brokers supply flexibility and opportunistic choices for users and foster the competition between Clouds.

2.2 Literature Review

In this section, we provide a comprehensive survey on the current literature on various Cloud Broker models. Our studies reveal that, most of the proposals on Cloud Broker architecture and mechanisms are found in the literature from the year 2009. In this literature review, we specifically focus on Cloud Brokers and hence Broker mechanisms in other related fields such as Grid Brokers are not discussed here. Taxonomy and classification of Grid Brokers can be found in [15], [16] and [17]. We have categorized the Cloud Broker models according to the services offered by them.

2.2.1 Cloud Service Arbitrage Models

Different from traditional arbitrage mechanisms which involves in simultaneous purchase and sale of an asset to make profit, a Cloud Service Arbitrage aims to enhance the flexibility and choices available for users with different requirements and to foster competition between CSPs. For example, a user may want to choose the best secure email provider whereas another user may want to choose the cheapest email service provider.

In [18], the authors propose a distributed negotiation mechanism wherein multiple buyers and sellers are allowed to negotiate with each other concurrently and an agent is allowed to decommit from an agreement at the cost of paying a penalty. This scheme is designed specifically for IaaS Clouds and proposes a dynamic pricing scheme. But the utility function can take into account only deadline and price.

In [19], the authors propose a Broker model to help the users to choose an appropriate IaaS CSP. Broker helps prospective Cloud consumers to compare and contrast different CSPs according to their specific requirements. It also helps them to test benchmark applications on different CSPs to get a better estimate of the cost and the performance. In [20], the authors propose a Broker based on Analytic hierarchy process (AHP) in which users can specify their requirements such as reliability and Broker displays a catalogue. Users choose appropriate IaaS CSP based on the catalogue. A basic implementation is done on Eucalyptus. But these schemes are user-centric and do not allow the CSPs to interact with Broker to adapt to market conditions.

In [21], the authors propose a scheme called as RAINBOW for choosing appropriate CSP based on deadline and budget constraints. The scheme has several limitations. It is designed for handling only compute resource requirements. Further CSPs are not allowed to use any dynamic pricing strategies and they can not adapt to market conditions using Broker. In [29], the authors propose a Cloud resource negotiation scheme to choose appropriate CSPs for Cloud users. Both schemes do not describe any system modeling or performance evaluation. In [22], the authors propose a Bayesian learning mechanism for resource allocation in

Cloud. Users submit their bid and auctioneer selects appropriate CSP after a set of auctions. Users use Bayesian learning mechanisms to update their bid prices whereas CSPs offer a fixed pricing mechanism. Main drawback is the assumption that several users compete for same resources which may not be the case in reality.

In [23], the authors propose an intelligent Cloud resource allocation scheme which helps in discovering all available resource configurations, choosing the desired configuration, negotiating an SLA, monitoring the SLA and assisting in the migration of services between CSPs. But only single resource requests are considered. Further, details of the pricing policies of CSPs are not presented. No other parameters (such as trust, security, geographical information etc) are considered in their modeling.

In [24], the authors present a market exchange framework based on auctions for resource negotiation in Cloud environments. But the model is restricted for bidding compute resources. They specify that it supports multiple types of auction mechanisms, but do not specify the one used in their performance evaluation. Further, their performance evaluation does not specify the relative merits/demerits of different auctions in their model for users and CSPs. The model considers only compute resources and seems to be applicable to any distributed systems such as Grid and failed to prove the advantages for CSPs to use this system.

In [25], the authors propose a market-oriented time optimization and cost optimization resource scheduling strategies for IaaS Clouds (offering compute resources) using budget and deadline constraints. But, the optimization is not done across multiple CSPs and CSPs offer fixed pricing. Further, this scheme

can be applied to any distributed environments such as Grids and the applicability in a Cloud context is not specified.

In [26], the authors present a scheme called as MC-QoSMS which collects QoS specifications from CSPs and QoS requirements from users and finds a suitable match based on reduct in rough set theory. The model considers various QoS parameters such as availability, security etc. But handling of QoS parameters are not modeled realistically. Further, this model does not allow CSPs to adapt to market conditions.

In [27], the authors propose an architecture and a scheme for a Cloud Coordinate to improve the performance of various entities in a Cloud ecosystem. They model a market which trades VMs. Cloud Exchange is presented to help in negotiation services and a Cloud Coordinator is available for SOA implementation. But, a market competition is not created and CSPs can not adapt to market conditions to improve their revenue. Further this scheme is designed to handle only compute resources.

In [28], the authors propose a knowledge-based auction model for trading resources in Cloud environments particularly for *futures market* and *spot market*. The intersection of demand curve and supply curve is considered as the market clearing price at which all resources are sold. The model is very limited as it is restricted to handle the limited supply resources (Future Market and Spot Market). Hence it is applicable to any other distributed domain such as Grid.

A comparison of all the Cloud Arbitrage mechanisms described in this section are summarized in Table 2.1.

2.2.2 Cloud Service Aggregation Models

Cloud Aggregation Brokers help to deploy customer applications across multiple CSPs for various reasons. Users may have some deadline and budget requirements which needs the application to be distributed across multiple CSPs, or users may require a service which can be obtained only by combining services from multiple CSPs. The second category is also known as *Cloud Broker Integration*.

In [30], the authors propose a distributed resource management scheme based on repeated non-cooperative and cooperative games in a dynamic federation platform for data intensive applications. In their pricing scheme, the price offered is based on the revenue obtained. But the model arises convergence issues due to the usage of repeated games and the utility function can handle only price and social welfare.

In [31], the authors present an architecture for Cloud aggregation and bursting for object based sharable environment specifically for IaaS storage Clouds. In [32], the authors propose a price based Mixed Integer program for splitting the load among CSPs. In [33], the authors propose a social network based approach for provisioning and managing Cloud resources using the principles of peer-to-peer networking and utility computing model. In [34], the authors propose an architectural strategy for the provisioning and delivery of services in Cloud communities, ecosystems and business networks. All these models only propose the Broker architecture and lacks in giving a detailed system modeling and performance evaluation.

In [35], the authors propose a genetic algorithm for estimating suboptimal sets of

resources and an agent-based approach for executing bag-of-tasks (BoT) applications simultaneously having budget and deadline constraints. But the solutions found are suboptimal and the model is not capable of preempting tasks when required. Further, the model does not take into account the cost involved in data communication and computation. In [36], the authors present a security enhanced coordinator to establish SLA to ensure QoS. It also monitors the work load and triggers on demand resource provisioning when required. But the scheme is designed only for real-time online interactive applications such as online games.

In [37], the authors propose a Cloud Brokering architecture which optimizes the placement of virtual infrastructures across multiple CSPs and also abstracts the deployment and management of infrastructure components in these CSPs. In [38], the authors propose a binary integer program problem for multi-provider hybrid Cloud setting based on deadline and budget constraints. These two schemes consider only IaaS compute Clouds and the problems are shown to be NP-Hard. In the second case, there are cases wherein the problem solve time exceeded even 5 days without getting an optimal solution for hybrid Cloud setting. In [39], the authors propose a modular Broker architecture for optimal service deployments in dynamic pricing multi-Cloud environments. But they do not specify the time complexity for solving various optimization problems. Moreover, performance evaluation is restricted to one CSP (Amazon EC2).

In [40], the authors propose a Broker architecture *Uni4Cloud* to automate deployment and configuration of multi-cloud applications. It handles lock-in using standards such as OCCI [41] and OVF [42]. This architecture can be used only with those CSPs who follow OCCI or OVF standards. But this scheme lacks sup-

port for various features such as SLA monitoring, strategy to choose appropriate CSP based on user requirements etc. Moreover, only a case study is presented which is not sufficient to understand the behavior of the scheme in general cases.

In [43], the authors propose a Cloud architecture called *OPTIMIS* which has a Broker model as part of the Cloud delivery architecture. They focus on Broker integration service for integrating various CSPs and gives a detailed architectural overview. In [44], the authors propose a Broker architecture to support integration, delivery and management of composite services in a multi-provider heterogeneous networks environment. In both cases, only a prototype architecture is proposed with some details on different use cases. The do not provide the required details such as system modeling, implementation and/or performance evaluation.

A comparison of all the Cloud Aggregation strategies described in this section are summarized in Table 2.2.

2.2.3 Cloud Service Intermediation

Intermediation Brokers customize and build add-on services on top of Cloud services to incorporate additional features on current Cloud services. It may include services to enhance the user experience with Clouds such as add-on security features (e.g. Single Sign-On), integrated billing and monitoring services, financial services etc.

In [45], the authors propose an identity brokerage and federation model to facilitate distributed access, Licence management and SLA management for SaaS

and IaaS Clouds. In [46], the authors propose a change management approach for Cloud backed business process models using an intelligent Broker. In [47], the authors propose a Broker for optimal placement of VMs on Clouds. In [48], the authors propose an architecture for managing CSPs and users for specifying system decisions using rules and policies. But none of these scheme gives a detailed system model , implementation and/or performance evaluation.

There are few models that are designed for specific Applications. In [49], the authors specify a Broker for finding optimal route between Broker and the Cloud. In [50], the authors specify a Broker for monitoring and managing SLA between users and CSPs. In [51], a Broker-based trust model for identity federation (specifically Single Sign-On) in Cloud is proposed. In [52], coordination of cloud services, based on a tuple-space architecture is proposed to enable the CSPs to advertise their capabilities.

In [53], an optimization problem has been formulated to maximize the successful allocation with QoS parameters such as bandwidth, data size and time for reserving and provisioning resources for data-intensive Cloud applications. But, the optimization problem formulated is not solved to obtain optimal values for the problem considered. Further, the problem considered is user-centric and CSPs can not adapt to market conditions. Finally, a limited application scenario is described based on IPTV.

In [54], the authors propose an options market for purchasing long-term contracts based on history and helps CSPs for resource forecasting. But, it does not specify how to deal with multiple CSPs and only compute resources are considered. In [55], a CloudBank has been proposed to provide analysis and guidance on price

offers by various CSPs to users. The scheme considers only in the formulation of price based on historical usage of resources and lacks in considering various Cloud parameters such as trust, security and reputation. Performance evaluation is not clear and the outcome of the experimental realization is not described clearly.

A comparison of all the Cloud Broker Intermediation strategies described in this section are summarized in Table 2.3.

2.3 Cloud Service Broker System Architecture

Our model consists of N users $\{C_1, C_2, .., C_N\}$ and M CSPs $\{P_1, P_2, .., P_M\}$ which are connected together through a Broker. Broker maintains few databases about the current system to aid users and CSPs to make their business decisions. A detailed architecture for the Broker is illustrated in Figure 2.2. This Broker architecture consists of three major components as described below.

2.3.1 Job Distribution Manager (JDM)

Job Distribution Manager is responsible for receiving User's job requirements, choose appropriate CSP selection strategy, informing appropriate CSPs about the jobs, and maintaining the job distribution statistics. When a user submits job requirements, the *Job Classification* module analyzes the job requirements and decides the preferred CSP selection strategy (either Cloud Service Arbitrage or Cloud Aggregation).

In this thesis, we discuss three Cloud Service Arbitrage mechanisms wherein the users and CSPs behaves according to market situation to maximize their

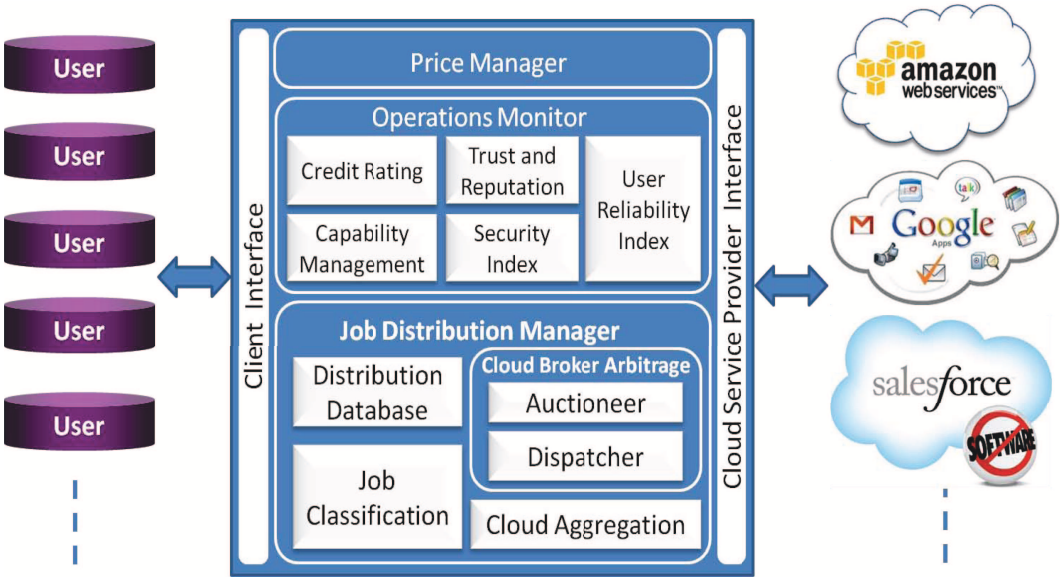


Figure 2.2: Architecture of the Proposed Multiple-Cloud Orchestration Mechanisms

corresponding utilities. If the policy is Auction-based mechanism, then the *Auctioneer* module will control the auction process and decides the winners. If the user wants aggregate its application across multiple Clouds, then the *Cloud Aggregation* unit will perform the necessary action. In all schemes, the *Dispatcher* module dispatches the job to corresponding CSPs after the CSP selection process is completed.

Further, the *Distribution Database* maintains a database about the job distribution statistics such as the winning CSP. This helps both CSPs and users to analyze their performance in the past with respect to other competing players in the market. For choosing appropriate CSPs and consolidating price information, JDM communicates with other components in the Broker.

2.3.2 Operations Monitor (OM)

Operations Monitor (OM) monitors, manages and maintain various information pertaining to both users and CSPs. The *Capability Management* module maintains databases about different resources and services offered by various CSPs. It also updates this information periodically when it notices any changes in the services offered by existing CSPs or when new CSPs enter the market. JDM makes use of this information to short-list the list of CSPs for participating in the game whenever a new job request arrives at JDM.

The modules *Trust and Reputation* and *Security Index* maintains information about the Reputation and security values of various CSPs from time to time. These values are supplied to users when requested. Users, based on their preferences use these values to choose appropriate CSP. A cumulative credit value is also derived based on these indices and user feedback which is maintained by the *Credit Rating* module.

Moreover a user reliability index is maintained by the *User Reliability Index* module. This is derived based on the trustworthiness of the feedback received from the users. This value is used by CSPs in making their price offer and by other users in forming their utility functions.

2.3.3 Price Manager (PM)

The Price Manager (PM) maintains the price offers supplied by CSPs from time to time. It is also responsible to calculate the current market price for different resources. This is used by CSPs to adjust their price offers. It also maintains other

financial matters such as maintenance of integrated billing information which can collectively calculate, display and manage the billing information from all the CSPs for the users.

2.4 Chapter Summary

In this chapter, we proposed the need for a Cloud Broker to solve several challenges that are persisting in the Cloud environments and conducted a comprehensive classification of existing Cloud Broker mechanisms into three categories: Cloud Broker *Arbitrage*, *Aggregation* and *Intermediation*. We presented the relative merits/demerits of the existing schemes and proposed a detailed Cloud Broker Architecture to solve several key issues existing in the current Cloud Computing environments.

Table 2.1: Comparison of various Cloud Broker Arbitrage Mechanisms

Scheme	Basic Mechanism	Cloud Services	Pricing Model	Extra Features	Market adaptation for CSP	Applications	Limitations
NG [18]	Distributed Negotiation Mechanism	IaaS	Dynamic Pricing	No	Yes	General	Only resource negotiation and allocation is considered, Utility is only based on deadline and offer price
SCB [19]	Helps users to choose appropriate CSPs	IaaS	No	No	No	General	User-centric scheme CSPs are not interacting with the Broker to know the market conditions
AHPBroker [20]	A Broker based on Analytic hierarchy process (AHP)	IaaS	No	Reliability, Uptime	No	General	User-centric scheme, CSPs are not interacting with the Broker to know the market conditions
RAINBOW [21]	Deadline and Budget constrained optimization for Cloud	IaaS	No	No	No	Compute resources	Very limited because CSPs are not modeled to offer dynamic price offers and CSPs can not adapt to market situation. No system modeling or performance evaluation.
BNEA [22]	Bayesian learning mechanism for resource allocation in Cloud	SaaS, PaaS, IaaS	Auctions for user bid and fixed pricing for CSPs	No	No	Compete for same resources	All users has to compete for same type of resource (may not be applicable in a practical sense)
ICRAS [23]	An intelligent Cloud resource allocation scheme	IaaS	Dynamic pricing (But detailas are missing)	No	Yes	Only single resource requests	Details of CSP pricing policies are missing. Other parameters such as trust, security, geographical information etc are not considered
Mandi [24]	Market Exchange framework for Clouds	IaaS	Auction based pricing	No	Yes	Compute resources	Lacks details and relative merits/demerits of different auctions in performance evaluation The model can be used for any distributed systems such as Grid and failed to proove the advantages for CSPs to use this system
TCO [25]	Market oriented resource scheduling for IaaS	IaaS	No	No	No	Compute resources	Considers only one IaaS CSP for allocation, Optimization is not done across multiple CSPs. CSPs offer fixed pricing. This scheme is application any distributed environments such as Grids and the applicability in a Cloud context is not evaluated
MC-QoSMS [26]	Rough-set theory based QoS management for IaaS Clouds	IaaS	No	Security, Availability	No	General	Handling of QoS parameters are not modeled realistically. Does not allow CSPs to adapt to market conditions.
InterCloud [27]	An architecture and a scheme for a Cloud Coordinate to improve the performance of various entities in a Cloud ecosystem	IaaS	No	No	No	Compute resources	A market competition is not created and CSPs can not adapt to market conditions to improve their revenue etc.
CRA [28]	A Knowledge-based auction model for trading resources in Cloud environments	IaaS	Continuous Double Auctions	No	Yes	Limited supply market for compute and storage resources	Very limited model which is restricted to handle the limited supply (Future Market and Spot Market). Hence it is applicable to any other distributed domain such as Grid. The performance evaluation is restricted to analyzing the efficiency of the scheme with no comparative study with respect to other schemes
CloudAgency [29]	Cloud resource negotiation	SaaS, PaaS, IaaS	No	No	No	General	Only a framework is proposed, lacks modeling and performance evaluation

Table 2.2: Comparison of various Cloud Aggregation Models

Scheme	Basic Mechanism	Cloud Services	Pricing Model	Extra Features	User/CSP centric	Applications	Limitations
DFP [30]	Distributed Resource Management	IaaS, PaaS	revenue based pricing	No	Both	Data-intensive Applications	Repeated games requires convergence time, Utility is formed only based on offer price and social welfare
CAB [31]	An architecture for Cloud Bursting and Aggregation	IaaS (Storage)	No	Security	Only a basic architecture is proposed	Not mentioned	Only a basic architecture is proposed. No analysis and performance evaluation
CRS [32]	Price based Mixed Integer program for splitting the load among CSPs	IaaS	No	No	Only a basic architecture is proposed	Not mentioned	Only basic details of the model are specified. Optimization problem is not solved. No performance evaluation is done
GA-based RE [35]	A genetic algorithm to deploy BoT applications into multiple Clouds to satisfy budget and deadline constraints	IaaS	No	No	User centric	BoT applications	The solutions found are suboptimal. Not capable of preempting tasks when required. Does not take into account the cost involved in data communication and computation
CBA [37]	A Cloud Brokering architecture which optimizes placement of virtual infrastructures across multiple clouds and also abstracts the deployment and management of infrastructure components in these clouds.	IaaS (Compute)	No	No	User centric	Compute intensive applications	Formulated problem is shown to be NP-hard.
HICCAM [38]	A binary integer program problem for multi-provider hybrid Cloud setting	IaaS (Compute)	No	No	User centric	Compute intensive applications	In case of hybrid Cloud, there are cases wherein the problem solve time exceeded even 5 days without getting an optimal solution
Uni4Cloud [40]	Facilitating modeling, deployment and management of applications in multicloud environments	IaaS	No	No	User centric	Applications that follow the standards	Needs continuous monitoring, CSPs should follow OCCI/OVF standards. Lacks support for various features such as SLA monitoring. Only a case study is presented which is not sufficient to understand the general behavior.
SOFCloud [36]	A model to support the federation of multiple autonomous infrastructure providers to provide a scalable IT infrastructure for delivering ROIA application as QoS-assured services	SaaS, PaaS, IaaS	No	Security	User centric	Real time online interactive applications	Limited application scope. Suitable for real-time online interactive applications such as online games
OPTIMIS [43]	Cloud brokerage model as part of a Cloud delivery architecture	SaaS, PaaS, IaaS	Tiered pricing	Security, Risk	Both	Not mentioned	Only a prototype architecture is proposed with some details on different use cases. Lacks fine details such as system modeling and performance evaluation
OCCS [33]	Social network based approach for provisioning and managing Cloud resources	SaaS, PaaS, IaaS	No	No	User centric	Not mentioned	Only an architecture is proposed. The work lacks system modeling and performance evaluation
SDF [34]	Service Delivery Framework	SaaS, IaaS, PaaS	No	No	User centric	Not mentioned	Only a framework is proposed, lacks modeling and performance evaluation
OSD [39]	Modular broker architecture for optimal service deployments in dynamic pricing multicloud environments	IaaS	Yes, but but the instance prices are assumed to be known in advance	No	Both	Compute intensive applications	They model several optimization constraints based on user requirements but they specified that the optimization problem is solved using MINOS solver. But they do not specify the time complexity for solving for various cases. Performance evaluation is restricted to one CSP (Amazon EC2)

Table 2.3: Comparison of various Cloud Broker Intermediation Mechanisms

Scheme	Basic Mechanism	Supported Cloud Services	Pricing Model	Intermediation Services	Limitations
FinanceCloud [54]	Financial Markets	IaaS (Compute)	Monthly price offers	Finanical management and advanced resource reservation	Does not specify how to deal with multiple CSPs, Only compute resources are considered, Only models price
BEinGRID [45]	Identitiy Brokerage and Federation	SaaS, IaaS	No	Identity management	Only a framework is proposed, lacks modeling and performance evaluation
BCM [49]	Broker Cloud communication paradigm	IaaS (Compute)	No	Route Finding	A very specific usage of Broker to find optimal communication route
CloudSLA [50]	Automatic construction of SLAs between Cloud users and CSP	IaaS	No	SLA management	Only a problem is discussed and its possible implementation by a Cloud Broker
EventCB [46]	Change management approach for Cloud backed business process models,	Not mentioned	No	Business change management	Only a basic model is described. No architecture and other details provided
CAA [47]	A broker for optimal placement of VM instances on CSPs	IaaS	No	VM placement	Broker is part of a comprehensive Cloud architecture. Lacks details such as modeling and performance evaluation
IFB [51]	Broker-based trust model for identity federation in Cloud	SaaS, PaaS, IaaS	No	Identity management	A model which can handle only SSO management by the Broker and hence has limited scope
PBMOCSMA [48]	A policy based Cloud management architecture	IaaS (Compute)	Yes. But no details provided	Policy management	Only an architecture is proposed. The work lacks system modeling and performance evaluation
DCSC [52]	Coordination of cloud services, based on a tuple-space architecture	IaaS	No	Coordination management	Very limited scheme. Does not specify the parameters considered for the coordination clearly.
CISF and SRB [53]	Reserving and provisioning resources for data-intensive Cloud applications	IaaS	No	Resource reservation and management	Optimization problem is not solved to obtain optimal values. The problem considered is user-centric and CSPs can not adapt to market conditions. Limited application scenario is described based on IPTV
Cloud Bank [55]	A resource agency to provide analysis and guidance on price offers by various CSPs to users	SaaS, PaaS, IaaS	Based on historical resource usage	Price analyzer	The scheme considers only in the formulation of price based on historical usage of resources and lacks in considering various Cloud parameters such as trust, security and reputation. Performance evaluation is not clear and the outcome of the experimental realization is not described clearly.
SBMA [44]	A Broker architecture to support integration, delivery and management of composite services in a multi-provider heterogeneous networks environment	SaaS, PaaS, IaaS	No	Composite services - Integration, Delivery and Management	An archiecture is presented. But the work lacks in providing any modeling or performance evaluation details.

PART I: MULTIPLE CLOUD ARBITRAGE MECHANISMS

Chapter 3

Broker-based Cloud Service Arbitrage Mechanisms using Sealed-bid Double Auctions and Incentives

3.1 Introduction

Detailed Cloud Broker architecture and various Cloud Broker categories were described in the last chapter. In this chapter, we propose two *Cloud Service Arbitrage* mechanisms. The objectives of these mechanisms are to help the Cloud users choose appropriate CSPs and for the CSPs to give opportunistic choices and increase the mutual competition to improve their performance. We make use of some principles of game theory such as *Continuous Double Auctions*, *Incentives* and *Belief-based repeated games with imperfect private monitoring* to design var-

ious aspects of the Broker in terms of formulating the utility function, dynamic pricing strategies, user reliability etc.

3.2 Important Terms and Definitions

Initially, without any loss of generality, we assume that all CSPs have similar capabilities and are able to service any incoming job request. Later we show that, if CSPs offer multiple services, our schemes can handle it seamlessly. Important terms used in this chapter are described as follows:

User Return Ratio (RR_{ij})

User Return Ratio captures the influence of how often a user C_i gets its job serviced by a CSP P_j, after evaluating the quotes sent to C_i by all CSPs in the system. It may be noted that this ratio is actually a function of time as it has a strong bearing on the number of jobs (e_{ij}) from C_i executed by P_j with respect to the number of job requests (S_{ij}) submitted to P_j by C_i. Thus we define RR_{ij} as:

$$RR_{ij} = \frac{e_{ij}}{S_{ij}} \tag{3.1}$$

Expected Return Ratio (ER_{ij})

It denotes the number of times a CSP expects a user to get its job serviced by it, when all CSPs are treated fairly in the system. If M CSPs are present in the system, then ER_{ij} can be simply computed as the inverse of M, i.e., $ER_{ij} = \frac{1}{M}$.

CSP Affinity Index (Y_{ij})

The affinity index ($Y_{ij} \in [0, 1]$) is a measure of the goodwill of any CSP P_j with any user C_i. There can be several contributors to goodwill in a real Cloud environment such as trust and reputation [56] for completing jobs without dropping them, job completion times (without delays), location of the CSP datacenter etc. Users can obtain this value in the form of *Credit Rating* from the *Operations Monitor* component of the Broker

CSP System Security (Z_{ij})

CSP System Security index ($Z_{ij} \in [0, 1]$) is the measure of the security level of a CSP P_j as perceived by a user C_i. Security is the assurance of secure computing services provided by system nodes and is calculated according to the brand image differential equation mechanism described in [57].

User Reliability Factor (χ_i)

Users submit feedback about CSPs after each transaction is completed. User Reliability Factor ($\chi_i \in [0, 1]$) is the measure of the trustworthiness of the feedback of a user C_i according to the Broker. χ_i decreases when C_i exhibits behavior that is not in correlation with the value maintained by the Broker.

3.3 Incentive-based Cloud Arbitrage Mechanism

In this section, we describe the incentive-based Cloud arbitrage mechanism. At any point of time, a user C_i can submit a job request with a demand D_i to a set of CSPs through the Broker. Each CSP P_j quotes an offer price to the Broker

and the Broker informs the collective offers from all CSPs to user C_i. The user, then calculates its utility as follows:

$$U_{ij} = \alpha((1 - X_{ij})) + \beta(Y_{ij}) + \gamma(Z_{ij}) \ \forall j \tag{3.2}$$

such that $\alpha + \beta + \gamma = 1; \ \alpha, \beta, \gamma \geq 0$

Then C_i selects the CSP which maximizes its utility. Here, X_{ij} is the normalized offered price[1], Y_{ij} is the affinity index and Z_{ij} is the security index. Based on individual requirements, different users may have different weightage for these factors. The job is submitted to the selected CSP, the corresponding databases are updated at the Broker and the RR_{ij} values are updated at the CSPs. This complete flow is summarized in Figure 3.1.

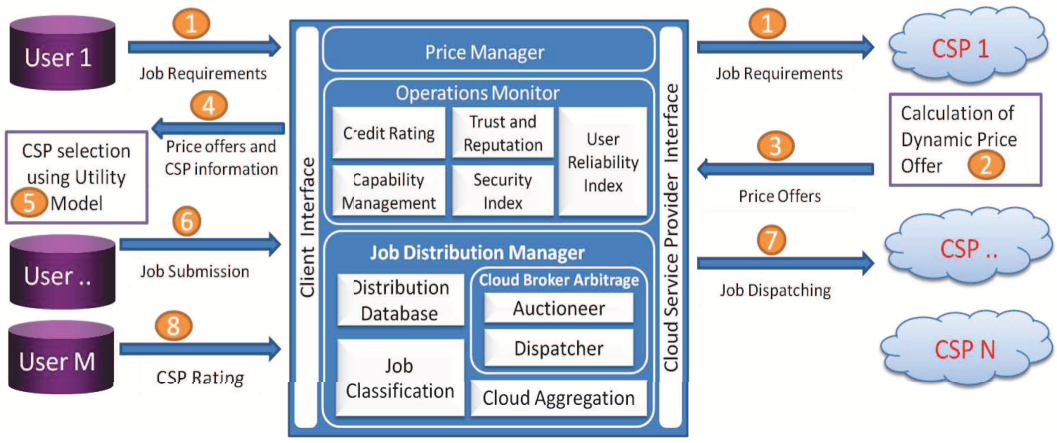


Figure 3.1: Flow Diagram for Incentive-based Scheme

[1]We normalize the price offers by various CSPs to make sure that the values lies between 0 and 1, with 1 being the highest price offered

Table 3.1: Incentive Scheme: Dynamic Pricing for CSPs

Condition	New Price Offer
$RR_{ij} > ER_{ij}$	$Pr_{ij}^{t} = Pr_{ij}^{t-1}$
$RR_{ij} < ER_{ij}$ and $L_j > \varphi$	$Pr_{ij}^{t} = Pr_{ij}^{t-1} + \Delta$
$RR_{ij} < ER_{ij}$ and $L_j < \varphi$	$Pr_{ij}^{t} = Pr_{ij}^{t-1} - \Delta$

3.3.1 Dynamic Pricing strategies for CSPs

In this scheme, CSPs use return ratio (RR_{ij}) to alter the price offers. In a practical scenario, the CSPs can consider any other parameters relevant to them for altering the price. If $RR_{ij} > ER_{ij}$, it implies that user C_i has been getting its jobs serviced by P_j more often than the expected average number of times and is a loyal customer. Hence P_j continues to offer the same price per resource Pr_{ij} to C_i. The offer price X_{ij} is calculated as $(D_i \times Pr_{ij})$, where D_i is the unit cost per resource for user C_i.

On the other hand, if $RR_{ij} < ER_{ij}$, there are two possible scenarios depending on the current load L_j for P_j at time t. If $L_j > \varphi$, (φ is some threshold value for the load in a CSP) it implies that P_j is popular with majority of the users and increases Pr_{ij} for C_i linearly by a small fixed amount Δ. If $L_j < \varphi$, it indicates a low popularity of P_j with the majority of users. So the CSP tries to recover by lowering its Pr_{ij} for C_i linearly by a small amount Δ, in an attempt to increase RR_{ij} and attract more users. Note that Pr_{ij} can not be lowered beyond a floor value. This is attributed to the fact that CSPs will not offer services which results in monetary loss for them.

3.3.2 Handling Security aspects by CSP

We define the weighted average H_j of the system security of CSP P_j as the weighted average of independent system security values Z_{ij} from all users and can be calculated as follows:

$$H_j = \frac{\sum_{i=1}^{N}(\chi_i \times Z_{ij})}{N} \tag{3.3}$$

The calculation of H_j takes into account the reliability of each user and their individual security value for the CSP P_j calculated by the user. It means that reviews submitted by reliable users are given more consideration compared to the reviews submitted by unreliable users. Initial value for the security index Z_{ij} for various CSPs can be calculated based on evaluating various security features supported by the corresponding CSPs. It can be calculated by using an assessment model to prioritize user's business systems and services for Cloud adoption [58]. Each time, user updates the security index as:

$$Z_{ij} = \eta(Z_{ij}) + (1 - \eta)(H_j) \tag{3.4}$$

such that $\eta \geq 0$

The value of η indicates the level of importance that the user gives to its own perception and the weighted average of fellow users' perceptions in the system. CSPs can modify their security features anytime by knowing user's perception of its security features by using the values of H_j from Broker's databases.

3.4 Auction-based Multiple-Cloud Orchestration Mechanism

In this section, we describe a Multiple-Cloud orchestration mechanism based on auctions. A classification of classic auction types are summarized in Figure 3.2 [2]. (A detailed overview of various auction types can be found in [59].) We employ sealed-bid Continuous Double Auction (CDA) in our model. In a CDA, both sellers and buyers submit their bid information to a third-party mediator (The Broker in our case). Sellers submit their price offers, buyers submit their bid-values and the mediator decides the winner(s) by matching these two values. The choice of bids reflect the user's strategic attempts to manipulate the selling price.

In our Auction-based Multiple-Cloud Orchestration mechanism, users submit a

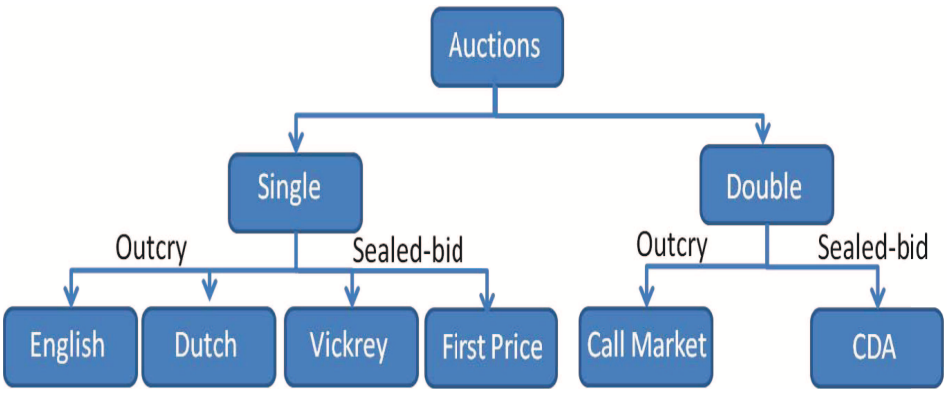


Figure 3.2: A classification of classic auction types [2]

user bid price for a resource, number of resources required and the desired CSP id to the Broker. The desired CSP P_j is selected by user C_i using the user's utility

function:

$$U_{ij} = \alpha T_{ij} + \beta R_{ij} \tag{3.5}$$

such that $\alpha + \beta = 1;\ \alpha, \beta \geq 0$

Here, T_{ij} is the trust value, R_{ij} is the reputation index and α and β are the weighing factors. The definition and calculation of T_{ij} and R_{ij} are described later in this section. Users obtain the required CSP information (such as reliability index) from the Broker prior to the calculation of the utility function. The Broker also maintains the minimum acceptable price offered by all the CSPs for the same resource in its *price manager* module. Using this information, the Broker can select the user bid which is the smallest value above the minimum value, as the winning bid for the auction. This complete flow is summarized in Figure 3.3.

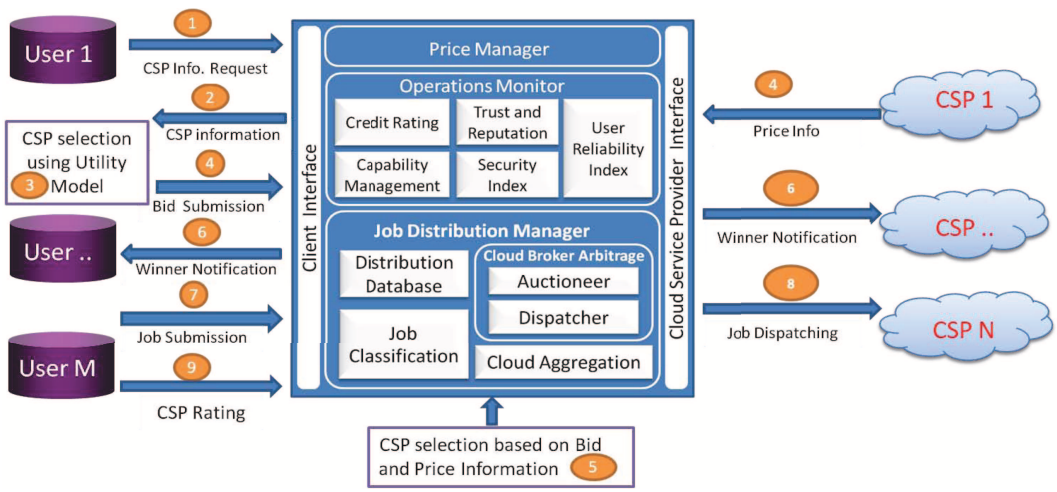


Figure 3.3: Flow Diagram for Auction-based Scheme

3.4.1 Pricing strategies for CSPs and Users

The CSPs learn market demand using the auctioning process through the Broker and alter their prices accordingly. The process of deciding prices is carried out at the beginning of every auction. CSPs keep track of the number of auctions in which they have not been picked. If this value exceeds a certain number, the CSPs attempt to put an end to the dry spell by lowering their offer price. If the CSPs have been picked, they raise their prices in the next auction. In order to facilitate this process, CSPs keep track of the Average Winning Price. Whenever the winning price is updated in the *Price Manager* module in the Broker, the CSPs modify their Average Winning Price to keep it up to date.

Meanwhile, users use this scheme and locate the vicinity of the minimum bid price. Users who mistakenly over-bid learn from the process and bid lesser in the next time (as the immediate previous winning bid is always available in the Broker). Apart from helping the users to reduce expenditure, this scheme also ensures that other users are protected from an unreasonable inflation of prices in the ecosystem. Users who mistakenly under-bid learn from the process and raise their bids in order to win the next time. Thus, the auction process creates a *win-win* situation to both CSPs and users in the system.

3.4.2 Calculation of Reputation by the Broker

Users select the desirable CSP using the *Reputation index* maintained by the *Operations Monitor* module of the Broker. Mathematically, the Reputation value is a combination of the performance reviews submitted by the users at the end of each auction and the minimum bid values submitted by the CSPs. The Reputa-

tion index R_{ij} for a CSP P_j after K auctions can be calculated as:

$$R_{ij} = \frac{\sum_{k=1}^{K} \frac{\sum_{i=1}^{N} Y_{ij}^{k} \chi_{i}^{k}}{N}}{k} + \frac{1}{minX_{ij}} \tag{3.6}$$

where first term is the average value of the product of the feedback submitted by the users about a particular CSP under consideration and the reliability index of those users calculated for the last K auctions. It means that reviews submitted by reliable users are given more consideration compared to the reviews submitted by unreliable users. The second parameter is the inverse of the minimum bid value from K auctions submitted by the corresponding CSP. If a CSP believes that its performance is rendering its reputation value too low, it can increase its reputation by lowering the prices upto some extent.

3.4.3 Calculation of Trust by the User

In addition to the reputation index, a *Trust* value (T_{ij}) is calculated by the users for each CSP. After each auction, T_{ij} is computed by C_i for P_j using the following relationship:

$$T_{ij} = \frac{\sum_{k=1}^{K} Y_{ij}}{k} \tag{3.7}$$

T_{ij} is the average value of C_i's affinity index to P_j for past k auctions for the same user. The users do not communicate this value between each other (assuming sealed-bid double auction). Further, T_{ij} is strictly a function of the feedback by individual users observed at the end of the auction.

3.5 Belief-based Game-theoretic Model for User Reliability

In order to enforce truthful feedback by the users after each auction, the *Operations Monitor* module of the Broker maintains a reliability index χ_i^t for each user C_i. Users are not aware of the reliability value assigned to them by the Broker. In each auction, the average performance of a particular CSP is calculated. If a user reports a performance value that is different from the average value beyond a threshold value, that user's review is viewed as a special case. If this user consistently submits such reviews, that user's reliability is lowered.

We update the reliability value in the *Operations Monitor* based on imperfect private monitoring (using belief-based non-cooperative repeated games) [60]. Let c be correct feedback[1] submitted by the user and d be the defective feedback submitted by the user at time t. Further assume $z \in [0, 1]$ to be a very small value indicating the noise in the observed feedback. A noise in this case shall be the number of extra chances that the user gets before the user has been marked as a completely unreliable user. Small value of noise (closer to 0) indicates more extra chances given to users to correct themselves and high value of noise (closer to 1) means Broker gives lesser number of chances to the user when defective feedback is submitted and marks the user as unreliable quickly. Then, Broker updates the value of $\chi_i^t : [0, 1] \rightarrow [0, 1]$ after each iteration [60] based on the feedback received from the users. When a correct feedback c is received, χ_i^t can be calculated as

[1] In a *correct feedback*, the performance value reported by the user will not deviate from the average performance value calculated by the Broker above a threshold value θ. Otherwise, the feedback will be considered as a *defective feedback*.

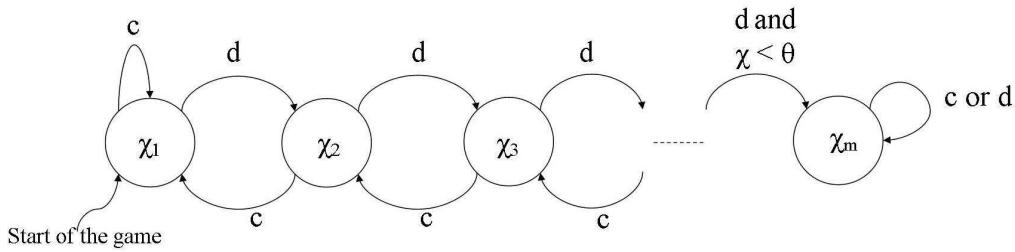


Figure 3.4: The state transition diagram for calculating the reliability index

[60]:

$$\chi_i^t = \frac{\chi_i^{t-1}(1-z)}{\chi_i^{t-1}(1-z) + z(1-\chi_i^{t-1})} \tag{3.8}$$

and when a defective feedback d is received, χ_i^t is:

$$\chi_i^t = \frac{\chi_i^{t-1}z}{\chi_i^{t-1}z + (1-z)(1-\chi_i^{t-1})} \tag{3.9}$$

We can observe that, if the initial value of χ_i^t is 1, then χ_i^t will be close to 1 as long as a user submits correct feedback (following Eq. 3.8). If the user submits a defective feedback, his reliability index χ_i^t will go down based on the calculation of Eq. 3.9. The calculation of χ_i^t can be represented using a state transition diagram as illustrated in Figure 3.4. In the state diagram, χ_1 is the reliability value for a user who always submits correct feedback. When the reliability index goes below a threshold θ, the user reliability index becomes χ_m and that user is marked as an unreliable user. Thus we can observe that $\chi_1 > \chi_2 > \chi_3 > ... > \chi_{m-1} > \theta > \chi_m$.

3.6 Performance Evaluation

In this section, we conduct extensive performance evaluation studies to analyze the *Revenue* obtained by the CSPs and the *Expenditure* incurred by the users,

in our schemes. We consider the following cases as most important issues to be validated via performance evaluation studies in the context of our study:

- Comparison of the revenues obtained in various cases
- Effect of user preferences in the utility function
- Effect of CSP preferences in the CSP revenue
- User migration between Auction based and Incentive based models
- Cloud market offering multiple services

The performance evaluation parameters are summarized in Table 3.2.

Table 3.2: General performance evaluation parameters

COMMON PARAMETERS	
Number of users	1000
Number of CSPs	10
Number of iterations	50000
Minimum Price	50
Maximum price	70
Plot value iteration	Every 10000
Threshold value for load in CSP (φ)	0.5

3.6.1 Comparison of the revenues obtained in various cases

Initially, we conduct experiments to study the revenue obtained by the CSPs in a variety of situations. We consider various budget requirements by users in Auction-based scheme. We define a metric Hit-ratio as the ratio of the number of successful auctions (A_s) to the total number of auctions (A_t). Mathematically, Hit Ratio $= \frac{A_s}{A_t}$. As the users increase their minimum bid value, the hit-ratio also increase. We also consider the revenue obtained in Incentive-based scheme and

the original scheme (A scheme without the presence of a Broker). All parameters in the utility function are given equal weightage in both schemes and the results are plotted in Figure 3.5.

We can observe that both Incentive-based and Auction-based schemes result in a fair revenue for all CSPs offering same resources compared to the traditional Cloud system without Broker. This is further corroborated by Jain's fairness index[1] [61] plotted for the revenue obtained by the CSPs, in Figure 3.6. We calculate Jain's fairness index as:

$$F(I_1, I_2, ..., I_N) = \frac{(\sum_{j=1}^{N} I_j)^2}{N.\sum_{j=1}^{N} I_j^2} \tag{3.10}$$

where, I_j is the gross income for CSP P_j. We can see that fairness index is close

[1]This index rates the fairness of a set of values. In our model, we calculate this index in choosing various CSPs by the users. A value close to 1 indicates that the system treats all CSPs fairly.

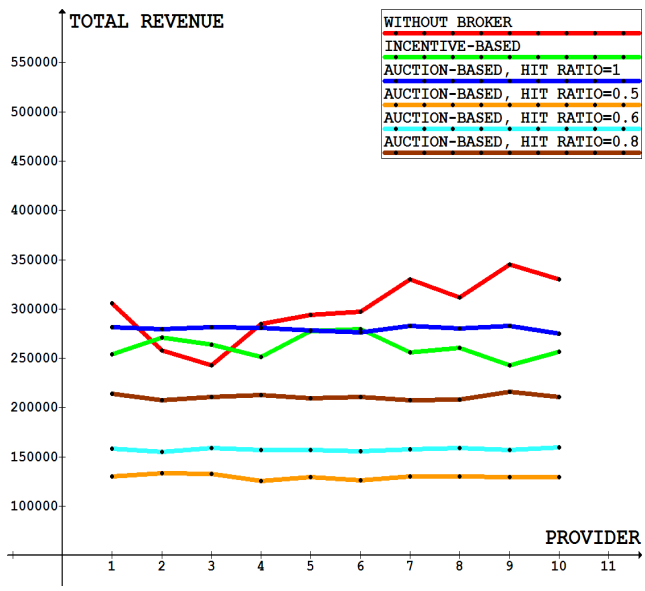


Figure 3.5: Comparison of revenue obtained in different cases

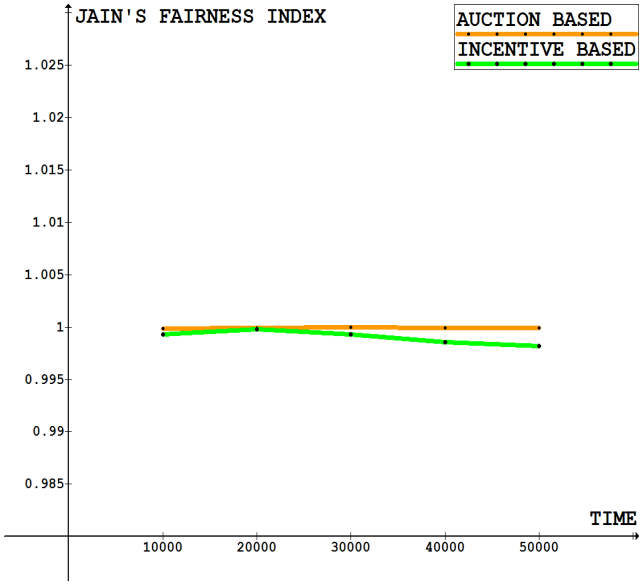


Figure 3.6: Jain's Fairness Index

to unity in both models. Further, we can observe that, in the scheme without Broker, users have limited knowledge about CSPs. They only have their own perception about CSPs and they make decision. Hence, some CSPs are chosen more often compared to others. With our model in place, users can choose the best CSP which matches all their requirements such as budget, security features, reputation etc. When the minimum budget is set to be the same in all models, Incentive-based model results in a higher *Revenue* for CSPs whereas Auction-based model results in about half of that (Auction-based with hit-ratio=0.5 in Figure 3.5). This is because, in Auction-based model, some of the auctions are not successful because of the higher bid price quoted by the CSP as compared to user's budget specifications. When users are willing to pay more, the hit ratio also increases and finally achieves a hit ratio of 1 where, the revenue achieved by the CSPs are comparable to other models.

3.6.2 Effect of user preferences in the utility function

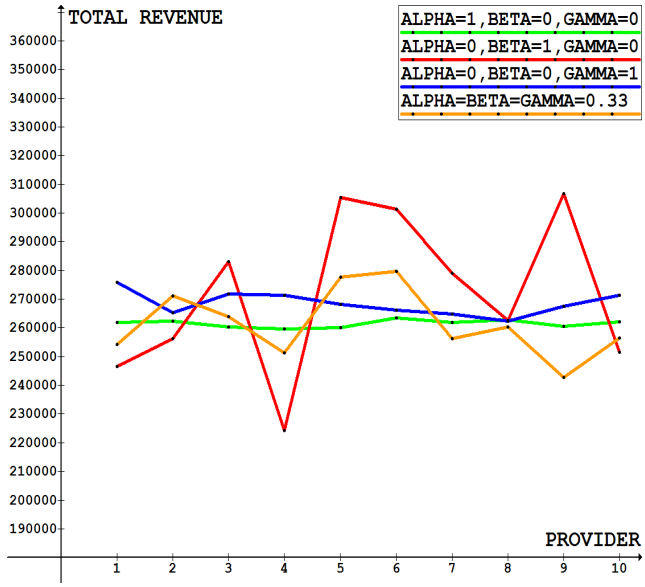


Figure 3.7: Effect of user preferences in Incentive-Based model

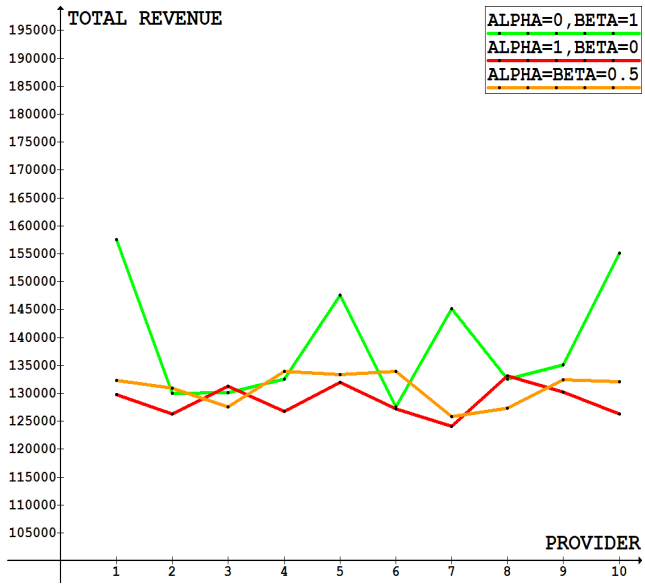


Figure 3.8: Effect of user preferences in Auction-Based model

Now, we analyze the effect of user's preferences in their utility function in

choosing the CSPs. For this purpose, we vary various weightage values in Equations (3.2) and (3.5) for Incentive-based and Auction-based mechanisms respectively and the results are plotted in Figures 3.7 and 3.8 respectively.

We can observe that, when the users give equal weightage to all parameters in the utility function, the CSPs are treated fairly and they obtain a fair amount of revenue. Whereas when the users give more importance to only trust and reputation (in Auction-based scheme) or only security index or affinity index (in Incentive-based scheme), the total revenue obtained by different CSPs vary. This is because some CSPs have a higher value for these parameters compared to others. For example, in Figure 3.8, CSP 1 and CSP 10 obtain a large revenue compared to other CSPs when $\alpha = 0$ and $\beta = 1$. This is because, these two CSPs maintain higher reputation values compared to others.

3.6.3 Effect of CSP preferences to participate in the proposed schemes

In this subsection, we analyze the effect of CSP preferences to participate in the two proposed schemes. Initially all CSPs participate in the proposed schemes and later after 25000 iterations, 5 CSPs decided to leave the system and function independently. The revenue obtained by CSPs for Auction-based and Incentive-based schemes for 50000 iterations are plotted in Figures 3.9 and 3.10 respectively. Some users based on their individual perception, choose one of these independent CSPs whereas other users choose one of the CSPs participating in the proposed Broker-based mechanism based on the utility function. We can observe that those CSPs who continue to participate in the Broker-based scheme achieves a higher

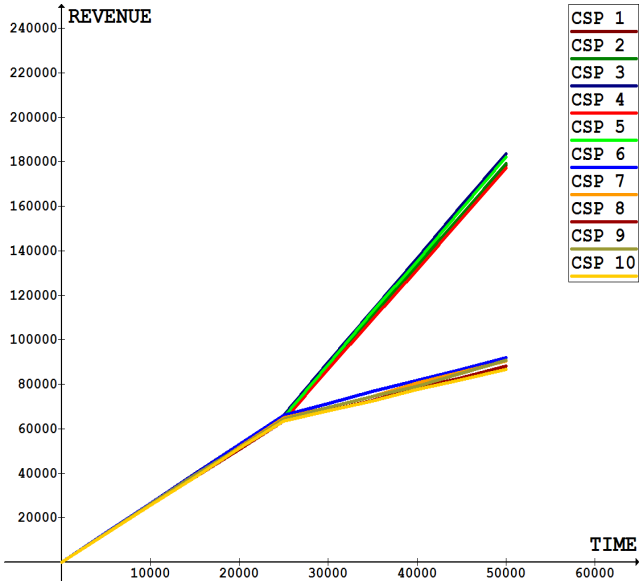


Figure 3.9: Effect of CSP preferences in Auction-Based model

revenue compared to those who leave the system. Further, CSPs participating in the Broker-based scheme obtain higher revenue as more users started choosing them based on the collective information obtained through the Broker.

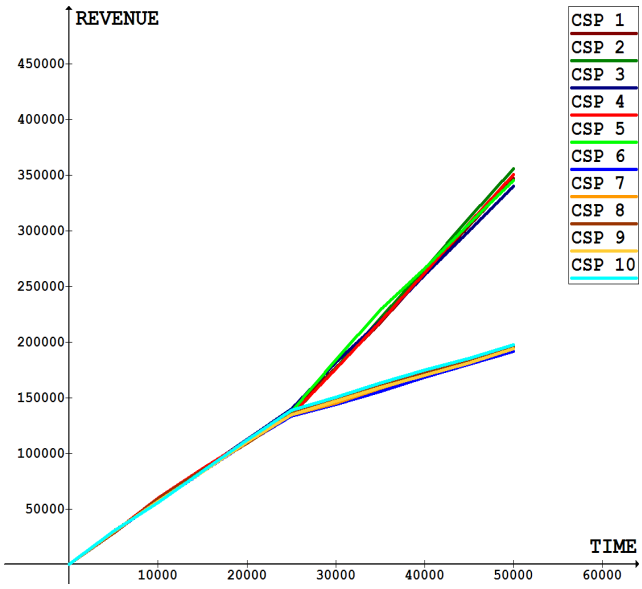


Figure 3.10: Effect of CSP preferences in Incentive-Based model

3.6.4 User migration between the proposed schemes

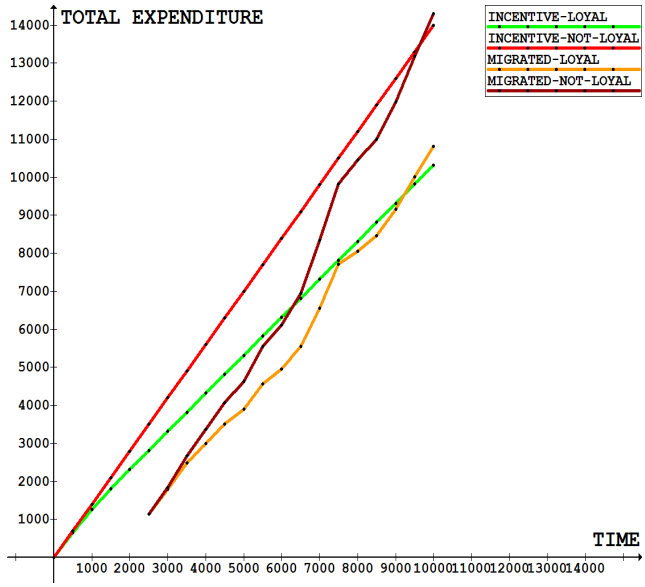


Figure 3.11: Migration from Auction-Based to Incentive-Based

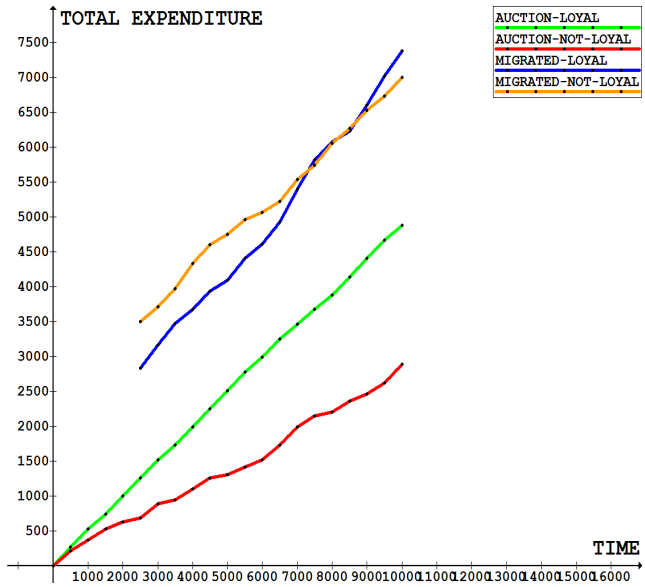


Figure 3.12: Migration from Incentive-Based to Auction-Based

Now we conduct experiments to analyze the average user expenditure when

some users migrate from Auction-Based model to Incentive-Based model and vice-versa. We conduct the performance evaluation for $10,000$ iterations with 100 users, and 50 users are migrated from one scheme to the other after 2500 iterations. These users have the knowledge about the whole system with respect to their parent scheme and now they choose the required CSP based on this historic information.

The user expenditure for migration from Auction-Based to Incentive-Based scheme (for one loyal user[1] and one not-loyal user chosen randomly) is plotted in Figure 3.11. A loyal user has less expenditure compared to a not-loyal user after few iterations. This is because, when a user is loyal, the CSP reduces or does not change its offer price for this user whereas if a user is not loyal, the CSP offers same or higher price compared to past offer price.

The user expenditure for migration from Incentive-Based to Auction-Based scheme is plotted in Figure 3.12. Comparing Figures 3.11 and 3.12, we see that initial expenditure is higher in Incentive-based scheme because the user has no option to specify a minimum budget requirement. When these users migrate to Auction-based scheme, both loyal and not-loyal users show similar trend in their expenditure as evident from Figure 3.12 and their expenditure converge after few iterations. This is due to the fact that, Auction-Based model treats all users (new or existing) in the same way. Further, winner of the bid and the winning price are decided by the Broker. Moreover the loyal-user has higher cumulative expenditure after few iterations because the user comes more often and uses resources for more time compared to non-loyal user and hence incurs higher monitory ex-

[1] A user is considered as a *loyal user* if he/she uses the same CSP very often for longer time

Table 3.3: Capability Management Database in the Broker

CSP-id	Services offered
CSP 1	Compute
CSP 2	Storage
CSP 3	Data
CSP 4	Compute, Storage
CSP 5	Storage, Data
CSP 6	Compute. Data
CSP 7	Compute, Storage, Data
CSP 8	Storage
CSP 9	Storage, Data
CSP 10	Compute

penses.

To summarize, if a user is loyal, and f a user wants to use the Cloud services more often, then it is better for him to follow Incentive-based scheme. Whereas if a user uses Cloud services infrequently, then the user may prefer Auction-based model. Further, in Incentive-based model, users can not specify their minimum budget and they choose the CSP who maximize his utility function. Whereas in Auction-based model, the users may reject some auctions, if the offered price is more than their bid price.

3.6.5 Cloud market offering multiple services

In real-world, CSPs often offer many kinds of services such as SaaS, PaaS and IaaS as discussed in Chapter 1. In order to demonstrate that our models can seamlessly incorporate CSPs offering multiple resources and users having diverse resource requirements, we consider CSPs offering up to three different services, Compute, Storage and Data[1] for the experiments in this section. The services offered by 10 different CSPs which are maintained by the *Capability Management*

[1]These services are chosen based on the services offered by Amazon Web Services (AWS) [12].

module, are summarized in Table 3.3. We also assume that users specify any of these services based on their requirements (which follows uniform distribution) when they submit the task. Here, the Broker sends the user requests to a subset of CSPs based on the type of services they offer.

Further, in the experiments, *compute* was requested more often compared to *storage* and *data*. So in the experiment, CSP 1 and 10 offers *compute* and generate higher revenue with respect to CSP 2 and CSP 8 which offers *storage* alone or CSP 3 which offers *data* alone. Due to the same reason, CSP 1 and 10 gets higher revenue compared CSP 5 which offers *storage* and *data*. Also note that, CSP 4 and CSP 6 generate higher revenue compared to CSP 1 because CSP 4 and CSP 6 offers two types of resources in which one type is *compute* resource. We conduct the experiments for 1000 iterations with 100 users and the results are plotted in Figure 3.13. We can observe that, CSPs offering multiple services obtain more

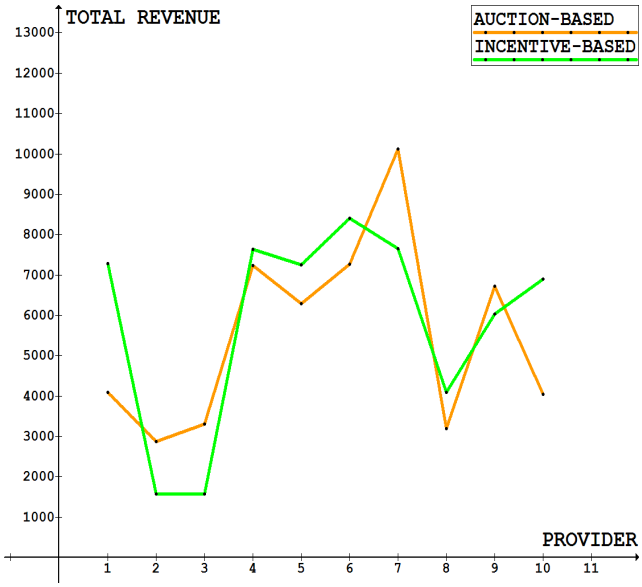


Figure 3.13: Revenue obtained when CSPs offer different products

revenue compared to CSPs offering single service. This is because they get more opportunities based on user requests. Further, it is interesting to note that CSPs offering same types of resources are treated fairly by both models. For example, CSP 5 and 9 (offering Storage and Data) gets similar revenues. This analysis reveals that our schemes can seamlessly handle CSPs offering multiple types of services and users with diverse requirements.

3.6.6 Remarks

- If CSPs offer multiple products/services, Broker can selectively handle user requests with a subset of appropriate CSPs without making any additional changes in the current schemes
- Our schemes can also seamlessly handle influx of new CSPs and users without affecting the existing market.
- It is interesting to note that users who use the Cloud services for a long time may prefer Incentive-based mechanism due to lower offered prices for the loyal customers and guaranteed win in single iteration. Whereas Auction-based scheme lowers user expenditure at the expense of response-time for choosing appropriate CSP (refer Figures 3.11 and 3.12).
- We can observe that the current Broker architecture already offers few light intermediation services such as maintaining security, reliability and reputation indices and market price information etc. More *Cloud Broker Intermediation* services can also be added for our schemes through the Broker very easily.

3.7 Chapter Summary

In this chapter, we proposed two schemes using incentives and continuous double auctions to connect users and CSPs through a Broker. The Broker maintains necessary information aiding the CSPs to offer attractive dynamic pricing policies to the users and, aiding users to choose appropriate CSP based on their requirements. Later we showed the effectiveness of our schemes through extensive performance evaluation studies and proved that our schemes give a fair treatment to all CSPs under certain conditions and always forces users and CSPs to reveal truthful information. Further, we showed the effect of user migration from one scheme to the other. We also showed the suitability of our schemes for different types of users, i.e. if a user wants to use the Cloud services more often, then it is better for him to follow Incentive-based scheme. Whereas if a user uses Cloud services infrequently, then the user may prefer Auction-based model. Finally, we showed that our schemes can seamlessly handle CSPs offering multiple types of resources and users having diverse requirements.

Chapter 4

Risk-aware Multiple Cloud Orchestration Mechanism

4.1 Introduction

In the last chapter, we described two Cloud Broker Arbitrage mechanisms one based on incentives and another one based on sealed-bid double auctions which can take into account various requirements such as budget requirements, trust, reputation etc. In this chapter, we model the utility function using the principles of Von Neumann-Morgenstern utility theorem (VNM-UT) [62] which can effectively handle user's risk bias towards the selection of appropriate CSPs based on their trust and reputation values. Similar to the other two Broker arbitrage mechanisms, this scheme also enables CSPs to adapt to market conditions to offer dynamic pricing strategies based on their history as well as the current market situation.

4.2 The Proposed Risk-based Cloud Broker Arbitrage Mechanism

Similar to the two models we have described in Chapter 3, users and CSPs are connected using a Broker (based on the Broker architecture described in Chapter 2) in this scheme as well. Users submit a job request to the Broker and Broker forwards this request details to all CSPs connected to it. CSPs send back the price offers to the user through the Broker. User has a utility function which is based on various parameters such as trust, risk bias and the cost involved. User chooses one CSP which maximizes its utility. Without any loss of generality, we assume that all CSPs have similar capabilities and are able to service any incoming job request. Later we show that, if CSPs offer multiple services, our scheme can handle it seamlessly. This complete flow is summarized in Figure 4.1.

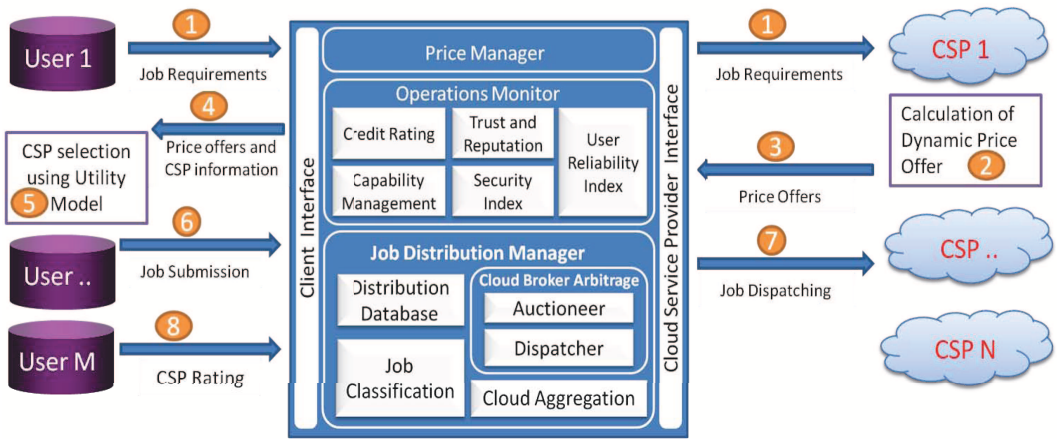


Figure 4.1: Flow Diagram for Risk-based Scheme

4.2.1 Formulation of Trust Function

A user makes use of a trust mechanism in which he/she calculates the trust using a reference scheme. A particular user uses the trust information from some of the other users to calculate its updated trust index (also called as Credit Rating) for each CSP. We propose two ways to calculate this rating. One based on trust without reference and the other one based on trust with reference.

Trust without Reference or *local Trust* is an accumulative value of *job rating*. To calculate *Trust with Reference*, user needs to select some users as reference users. In order to make sure that the selected reference users are not malicious, the user should have a rating for the reference users, which is referred to as *Reference Credit RC* in our system. With *Reference Credit* and *local Trust*, we can calculate *Trust with Reference*, which is used in the calculation of utility function.

Job Rating

After each transaction, user C_i who sends the request needs to rate the chosen CSP P_j's performance. The rating can be ranked according to user's expectations and is subjective. Some of the main factors in providing the rating could be, *meeting task deadline*, *adherence to Service Level Agreement (SLA) specifications* etc. In this chapter, we use $JR_{i,j})^n \in [0, 1]$ to represent user C_i 's Job Rating to CSP P_j in the nth transaction.

Local Trust

In our system, Local Trust (i.e. *Trust without Reference* or Job Rating) of user i to CSP j for all past transactions without reference is given by [63]:

$$TR_{i,j} = \frac{\sum_{n=1}^{f} \alpha^n JR_{i,j}^n}{\sum_{n=1}^{f} \alpha^n} \in [0,1] \tag{4.1}$$

where, f is the total number of past iterations considered in calculating the trust. It totally depends on the job Rating of all the past f transactions between User C_i itself and CSP P_j. In the above equation, $n = 1$ denotes the latest transaction and α^n in the equation means that transactions happened recently are given higher importance compared to transactions in the distant past. α is the decaying factor $(0 < \alpha \leq 1)$ and it determines the importance of the most recent transaction to the local trust.

Reference Credit

If a user C_i has never performed any transactions with CSP P_j in the past, it is necessary to have some other users' evaluation results as reference. It also helps the user to avoid bias judgement and to give more trustworthy evaluation of the CSP. However, it is necessary to rate the reference user γ so that the user can find the most appropriate reference user and avoid any malicious reference users. After n^{th} transaction, user C_i will compare its own job rating $JR_{i,j})^n$ to the offered trust $TR_{r,j})$ of reference user γ and rate the reference user γ accordingly as:

$$RC_{i,\gamma}^n = 1 - |JR_{i,j}^n - TR_{\gamma,j}| \tag{4.2}$$

Here, a value close to 1 means user C_i gives user j a better rating. User C_i's Reference Credit ($RC^n_{i,\gamma}$) to user γ will be accumulated to calculate an overall rating $RC_{i,\gamma}$ and it shows the trustworthiness of reference user γ according to user C_i. This value will be used in the calculation of *Trust with Reference*. $\beta \in (0, 1]$ is a decaying factor and it indicates the importance of most recent transaction to the credit of reference γ in user C_i's evaluation. Credit rating of user i to reference γ for all the reference actions is given by [63]:

$$RC_{i,\gamma} = \frac{\sum_{n=1}^{f} \beta^n RC^n_{i,j}}{\sum_{n=1}^{f} \beta^n} \tag{4.3}$$

Trust with reference

With both the local trust of user C_i to CSP P_j and the total value of f reference user's local trust to CSP P_j, we define *Trust with Reference* as [63]:

$$TR^r_{i,j} = \frac{TR_{i,j} + \sum_{n=1}^{f} RC^n_{i,\gamma} TR_{\gamma,j}}{1 + \sum_{n=1}^{f} RC^n_{i,\gamma}} \tag{4.4}$$

The local trust $TR_{i,j}$ of user C_i to CSP P_j is completely taken into account in the definition. The product of local trust $TR_{\gamma,j}$ of reference user γ to CSP P_j and the Reference Credit $RC_{i,\gamma}$ is used in the calculation of utility function

4.2.2 Formulation of User's Utility Function

We model the expected utility function using the principles of Von Neumann-Morgenstern utility theorem (VNM-UT) [62]. Let G be the set of all gambles (in our case, it is the set of CSPs in the market). Then a utility function formulated based on VNM-UT, satisfies the following axioms ([64], [62]):

- **Completeness:** If $g, g' \in G$, then either $g \succeq g'$ or $g' \succeq g$.
- **Transitivity:** For any three gambles $g, g', g'' \in G$, if $g \succeq g'$ and $g' \succeq g''$, then $g \succeq g''$.
- **Continuity:** For any three gambles $g, g', g'' \in G$ such that, $g \succeq g' \succeq g''$, then there exists some $\varpi \in [0, 1]$ such that $\varpi g + (1 - \varpi)g'' \sim g'$.
- **Independence:** For any three gambles $g, g', g'' \in G$ and $\varpi \in [0, 1]$, then $g \succeq g' \Leftrightarrow \varpi g + (1 - \varpi)g'' \succeq \varpi g' + (1 - \varpi)g''$.

Below we attempt to explain the physical significance of the above axioms. The *completeness* axiom says that, users have preference over all gambles and can rank them all. The *Transitivity* axiom says that, if g is preferred (or indifferent) to g', and g' is preferred (or indifferent) to g'', then g is preferred (or indifferent) to g''. An implication of the *continuity* axiom is that if g is preferred to g', then a gamble close to g (a short distance away in the direction of g'' for instance) is still preferred to g'. Finally, *independence* axiom says that, if users are indifferent between two possible outcomes, then they are indifferent between two gambles which offer them with equal probabilities, if the gambles are identical in every other way.

VNM-UT is widely used in literature ([65], [66], [67]) for modeling expected utility under various situations especially because, it can take into account user's risk-bias. In general, we can categorize users into risk-averse, risk-neutral or risk-seeking [64]. *Risk-Averse* users are reluctant to accept an offer with an uncertain payoff compared to an offer with a more certain, but possibly lower payoff. On the other hand, a *risk seeking* user prefers to take risk. Users who do not fall into

Table 4.1: Different types of users

User type	$v(TR^r_{i,j})$	$r(TR^r_{i,j})$
Risk averse	Concave	$r(TR^r_{i,j}) > 0$
Risk neutral	Linear	$r(TR^r_{i,j}) = 0$
Risk seeking	Convex	$r(TR^r_{i,j}) < 0$

either of these categories are *risk neutral* users.

Let $V(TR^r_{i,j})$ be a utility function following VNM-UT. A more concave nature of $V(TR^r_{i,j})$ implies more risk-aware users [62]. We can measure this mathematically using Arrow-Pratt measure of risk aversion $r(TR^r_{i,j})$. Mathematically,

$$r(TR^r_{i,j}) = -\frac{V'(TR^r_{i,j})}{V"(TR^r_{i,j})} \tag{4.5}$$

The nature of $V(TR^r_{i,j})$ and the range of values for $r(TR^r_{i,j})$ are summarized in Table 4.1. The utility function [67] can be defined as:

$$V(TR^r_{i,j}) = \frac{1 - e^{-\lambda TR^r_{i,j}}}{1 - e^{-\lambda}}; TR^r_{i,j} \in [0, 1], \lambda > 0 \tag{4.6}$$

Then,

$$V'(TR^r_{i,j}) = \frac{\lambda e^{-\lambda TR^r_{i,j}}}{1 - e^{-\lambda}} > 0 \tag{4.7}$$

,

$$V''(TR^r_{i,j}) = \frac{-\lambda^2 e^{-\lambda TR^r_{i,j}}}{1 - e^{-\lambda}} < 0 \tag{4.8}$$

and

$$r(TR^r_{i,j}) = \lambda > 0 \tag{4.9}$$

The explicit cost by the user with respect to each CSP can be expressed as:

$$E_c = \frac{P_t}{B} \tag{4.10}$$

where P_t is the price offered at time t and B is the budget specified by the user. Combining Equations (4.6) and (4.10), we formulate the final utility function for the user as:

$$U(TR^r_{i,j}) = \frac{1 - e^{-\lambda TR^r_{i,j}}}{1 - e^{-\lambda}} . \frac{1}{E_c} \tag{4.11}$$

Substituting the value of $TR^r_{i,j}$ and E_c from Equations (4.4) and (4.10) in (4.12), we get:

$$U(TR^r_{i,j}) = \frac{1 - e^{-\lambda(\frac{TR_{i,j} + \sum_{n=1}^{f} RC^n_{i,\gamma} TR_{\gamma,j}}{1 + \sum_{n=1}^{f} RC^n_{i,\gamma}})}}{1 - e^{-\lambda}} . \frac{B}{P_t} \tag{4.12}$$

The utility function can take into account risk, trust and cost parameters and the user chooses the CSP which maximizes the expected utility value.

4.2.3 Dynamic Pricing Strategies

In this section, we describe the formulation of dynamic pricing strategies for CSPs based on acceptance rate. We formulate the dynamic pricing model based on several factors such as the price offered in the last transaction, current market price maintained by the Broker and the acceptance rate of the CSP. Price offered at time t is given by:

$$P_t = P_{t-1} + (A_t - A_{thr})(\xi P_{t-1} + (1 - \xi) Q_{t-1}) \quad 0 \leq \xi \leq 1 \tag{4.13}$$

where, P_{t-1} is the price offered in the last transaction, A_t is the acceptance rate of this CSP at time t, A_{thr} is the expected acceptance rate of CSP, Q_{t-1} is the market price for this resource (i.e. average price of other CSPs offered this at time $t-1$) obtained from the Broker and ξ is the weighing parameter. The actual acceptance rate for a CSP at time t is given by:

$$A_t = \frac{\sum_{m=t_0}^{t} S_m}{\sum_{m=t_0}^{t} O_m} \tag{4.14}$$

where, $S(m)$ is the total resources sold from time t_0 till time t and $O(m)$ is the total resources offered from time t_0 till time t.

The CSP adjusts its price around the price it offered the last time to this user and the average price offered by all the CSPs Q_{t-1} according to the supply and demand of its offered resources. If $A_t > A_{thr}$, it means the CSP's offerings have been accepted adequately and it can choose to increase its price. If $A_t < A_{thr}$, it means the CSP's have been rejected too much and it should consider to reduce its price to win more users. ξ is the parameter indicating how much the CSP want to refer to its own price or the average price offered by all the users in last transaction.

4.3 Performance Evaluation

4.3.1 Simulation Setup

In this section, we conduct extensive performance evaluation studies to analyze the effectiveness of the proposed multiple Cloud orchestration mechanism. In our

simulation setup, there are 10 CSPs and 1000 users. $50,000$ requests are sent to the broker in each run. Table 4.2 summarizes values for various parameters used in our first set of performance evaluation experiments. Now, we describe the initialization of price and credit rating values and explain the rationale behind this initialization.

Table 4.2: General simulation parameters

Parameter	Values
Number of requests	50,000
Number of CSPs	10
Number of Users	1,000
Budget range of Users	[38, 140]

Initial values for price offers by various CSPs

In order to perform a realistic performance analysis of our model with a real Cloud environment, we use initial offered price of 10 CSPs from publicly available data [5]. We choose 10 CSPs who offer resources with same specifications (Table 4.3 summarizes the resource specifications of various CSPs) and use their price offers to initialize the price offers in our model. Table 4.4 specifies the initial price values of these 10 CSPs.

Table 4.3: Resource specifications of CSPs [5]

Resource	Specifications
RAM	8 GB
Storage	50 TB
CPU Power	$5x$
OS	Windows

Table 4.4: Initial price offers by various CSPs [5]

CSP id	CSP	Monthly Price (in dollars)
1	Bit refinery	38
2	CloudSigma	49
3	aTLanTIC.ner	45
4	OpSource	78
5	VPS NET	98
6	GoGRID	140
7	terremark	46
8	JoyentCloud	61
9	dediserve	112
10	AWS	123

Choice of values for Initial Credit Rating

Initially, we randomly generate 10 random values as the base credit for each of the CSP. Then, we generate the credit value of each user to each CSP based on the base credit. The generated credit value of each user to certain CSP is within 20% of the base credit of the CSP.

4.3.2 Effect of Dynamic Credit with static price

First we analyze the effect of our credit system. Keeping the price static, we repeat the experiments for static credit and dynamic credit and the results are plotted in Figure 4.2. In the case of dynamic credit, the trust (or credit rating) for CSPs by users changes with time based on their interaction and every time users use updated credit values in calculating the utility function.

Case 1: Static Credit: None of the users change their Credit Rating for the CSPs. From Figure 4.2, we can observe that, with static credit, only $CSP1$ and $CSP3$ are always chosen by the users.

Case 2: Dynamic Credit: In this case, $CSP2$ changes its trust features every

500 iterations until the credit reaches 0.9 and hence all users provide a higher credit rating to $CSP2$ by a small value. We can observe from Figure 4.2 that $CSP2$ obtains considerable gain in revenue by improving its credit rating obtained from the users.

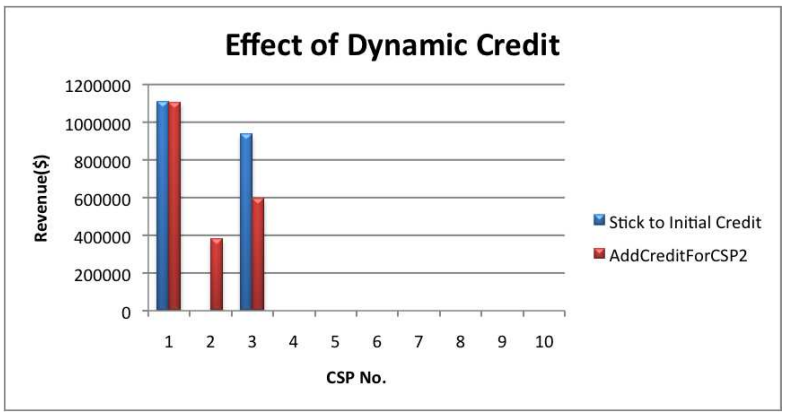


Figure 4.2: Effect of dynamic credit on CSP revenue

4.3.3 Effect of Dynamic Credit with dynamic pricing strategies

In this section, we conduct experiments to analyze the effect of dynamic credit with dynamic pricing strategies. As defined earlier, the credit rating of various users to certain CSP is around the base credit of the CSP. To show the effect of different credit, two credit settings are generated. In the first credit setting, base credits falls within the set $[0.2, 0.9]$ (case 1) and in the second credit setting, base credits are generated from the set $[0.6, 0.8]$ (case 2). Further, keeping the values of other parameters the same, dynamic pricing strategy is applied. In order to show the effect of credit, we change the price in a relatively large interval (Every 100 interactions). Simulation parameters used for the experiments in this section

are summarized in Table 4.5 and individual base credit values for various CSPs are summarized in Table 4.6. The results obtained in case 1 and case 2 are plotted in Figures 4.3 and 4.4, respectively.

5 Credit Ratings of CSP 1 is entirely different in two settings and we take this as

Table 4.5: Simulation parameters for Section 4.3.3

Parameter	Values
Number of requests	1,000
Number of CSPs	5
Number of Users	10
Minimum Budget	45
Maximum Budget	60
Minimum Price	40
Maximum Price	60

Table 4.6: Credit Setting 1

CSP id	Base Credit (case 1)	Base Credit (case 2)
1	0.217241	0.751348
2	0.746005	0.68454
3	0.487929	0.659554
4	0.758531	0.730438
5	0.761890	0.677734

a representative example to discuss our observations. In Figure 4.3, $CSP1$ does not generate revenue for first 500 interactions while in Figure 4.4, $CSP1$ begins to generate revenue immediately after 100 interactions and obtains relatively constant revenue after that. Further, we can observe that as all CSPs change their dynamic credit and price with time, they all obtain same cumulative revenue at the end of the experiments.

This reveals that if some CSPs have less credit rating (i.e. users do not trust them due to some reasons), by improving their trust features, they can get a higher rating among the users and hence obtain a higher revenue. In the absence of a Broker, it is difficult to adapt to market conditions such as changing the price

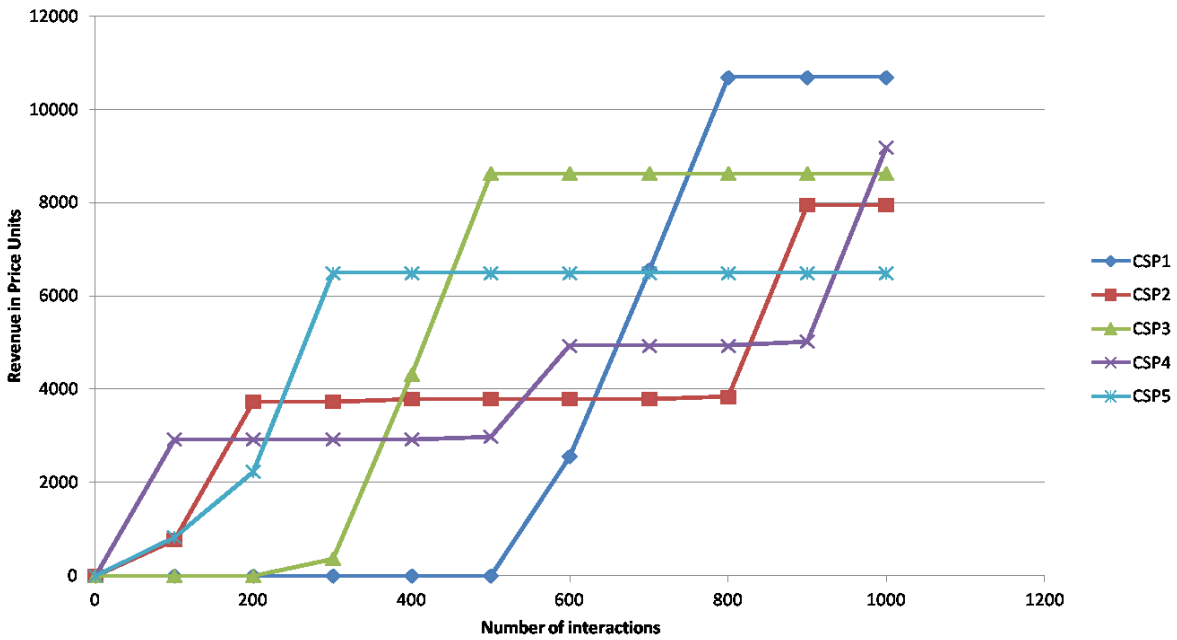


Figure 4.3: Effect of dynamic credit on CSP revenue for Setting 1

offers based on market price and improving the trust features to improve user credibility.

4.3.4 Analysis of Revenue for static and dynamic pricing cases

Now we analyze the effect of dynamic pricing strategies on the revenue of CSPs and compare it with the static pricing scenario. We set the risk parameter λ to 1 for all cases wherever it is applicable. The job rating of each user to each CSP is set to be varying within 20% of its initial credit. In case of dynamic pricing mechanism, we consider two cases with $\xi = 1$ (price offered depends only on the price offered by the same CSP in past) and $\xi = 0$ (price offered is changed based on both its past price and the current market price) and we also set the desired

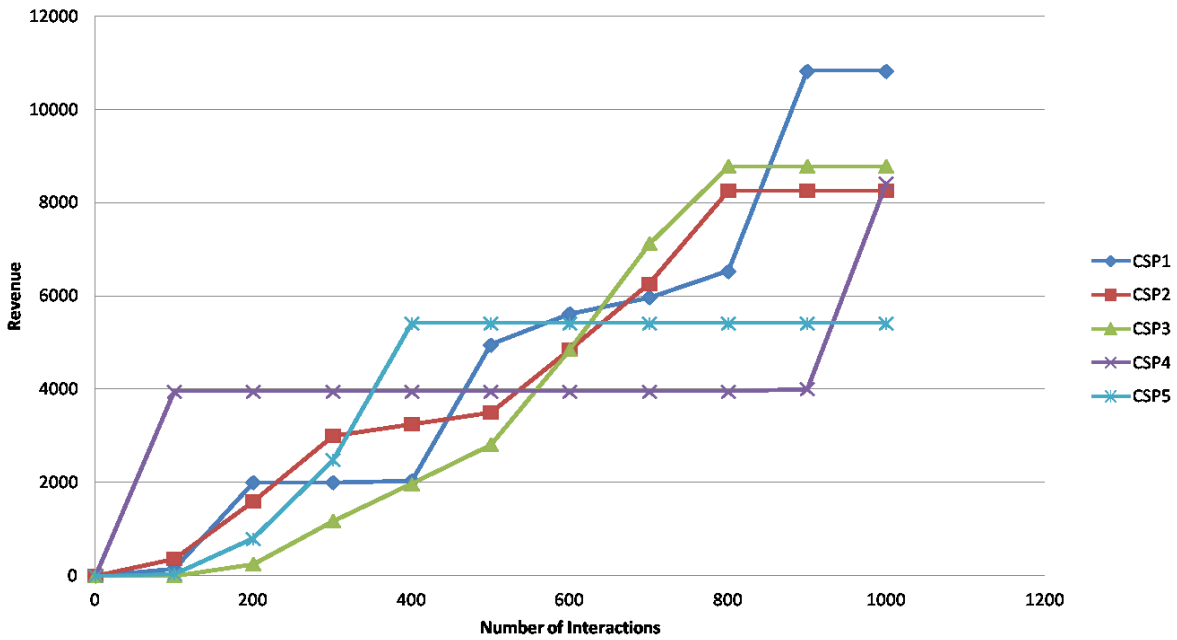


Figure 4.4: Effect of dynamic credit on CSP revenue for setting 2

acceptance rate for all CSPs to 0.1 for comparison purposes. We also compare the performance of the proposed schemes with current Cloud market wherein the CSPs are chosen according to user's own perception without any Broker. We term it as *Random* in our experiments. The results for total revenue and acceptance rate are plotted in Figures 4.5 and 4.6 respectively.

We can observe from the results that when CSPs offer static prices, only two out of 10 CSPs, which offers the least price are chosen every time and hence only two CSPs get the revenue. It shows that if a CSP can not respond to the market quickly, it can be an unfavorable choice and may lead to economic loss. After applying dynamic pricing strategy (in both cases), we can observe that all the CSPs generate revenue and they are treated more fairly in the market. Further, when $\xi = 0$, most of the CSPs generate higher revenue compared to the case with

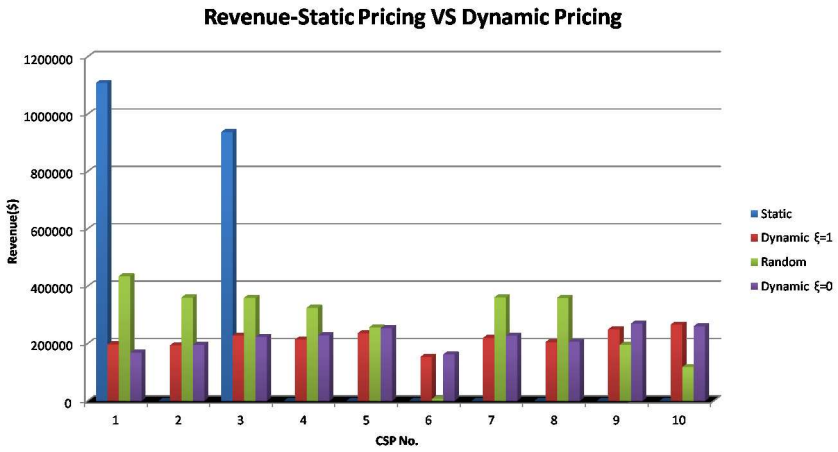


Figure 4.5: Analysis of revenue in static and dynamic cases

$\xi = 1$. This reflects that, CSPs are able to effectively make use of the market price information available in the Broker to increase their revenue and to adapt to the market conditions.

Further, we use Jain's fairness index to measure the fairness in revenue for 10 CSPs as done in Chapter 3. We record the total revenue obtained after every $10,000$ transaction for a total of $50,000$ transactions for different cases and the results are plotted in Figure 4.7. We can observe that, fairness index is close to 1 for our model with dynamic pricing strategies. In the beginning of the simulation, the values are smaller, but as the time progresses, CSPs adapt to market conditions by calculating and offering competitive prices and hence obtain fair revenue for all CSPs. We can also observe that, our dynamic schemes treat the CSPs fairly compared to the *Random* case where the CSPs are chosen by the users based on their own perception and understanding.

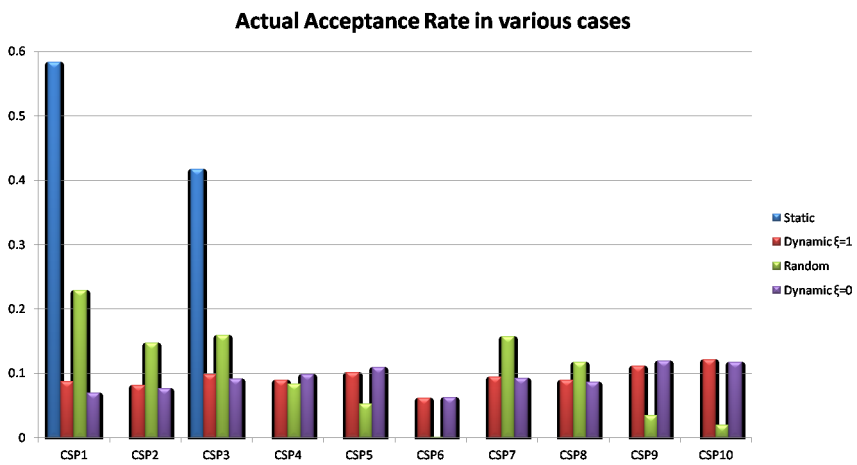


Figure 4.6: Acceptance rate for various CSPs

4.3.5 Analysis of various dynamic pricing mechanisms

Our price formulation expressed by Equation 4.13 has a variable parameter ξ which allows the CSPs to alter their price offer either based on its own past price, based on the market price or based on both of them. Here, we conduct experiments with $\xi = 0, 0.5$ and 1 and the results are plotted in Figures 4.8, 4.9 and 4.10 respectively. We conduct experiments for $50,000$ iterations and values are plotted after every $5,000$ iterations. The results are plotted for only $CSP1$ as a representative example.

When $\xi = 0$, the price adjustment fully rely on the average price of all CSPs. From Figure 4.8, we can observe that, $CSP1$'s offer price gets close to average price of all CSPs very quickly. When $\xi = 0.5$, the price adjustment depends on both CSP's own price and all other CSP's average price. From Figure 4.9, we can observe that, $CSP1$'s offer price still gets close to average price of all other CSPs but with a relatively low speed compared to the previous case. When $\xi = 1$, CSP adjusts its price offers fully according to its own offered price in the

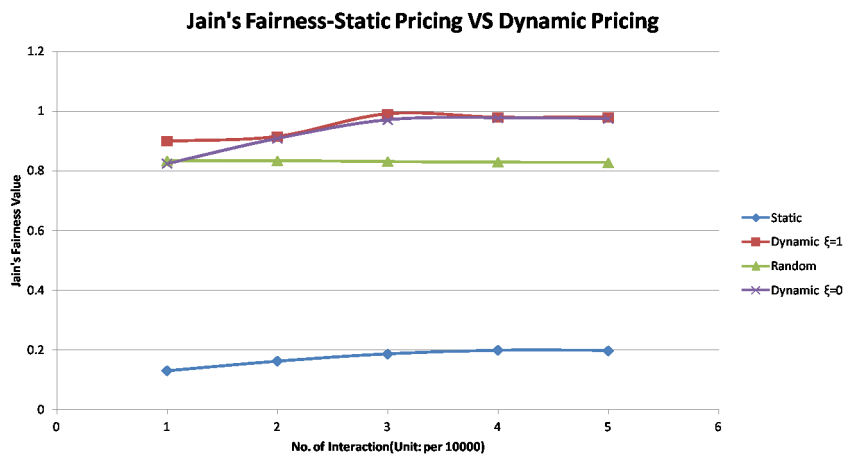


Figure 4.7: Analysis of Jain's Fairness Index for CSPs

last interaction and the CSP does not make use of the market price information available at the Broker. From Figure 4.10, it can be shown that $CSP1$'s price does not get close to the average market price.

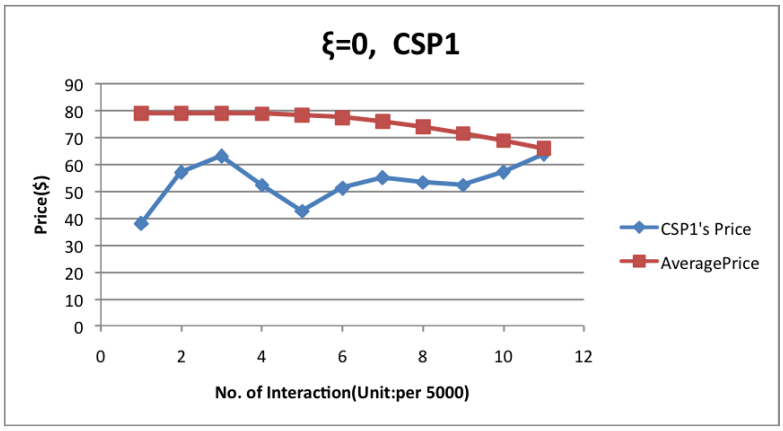


Figure 4.8: $\xi = 0$: Price adjustment only based on market price

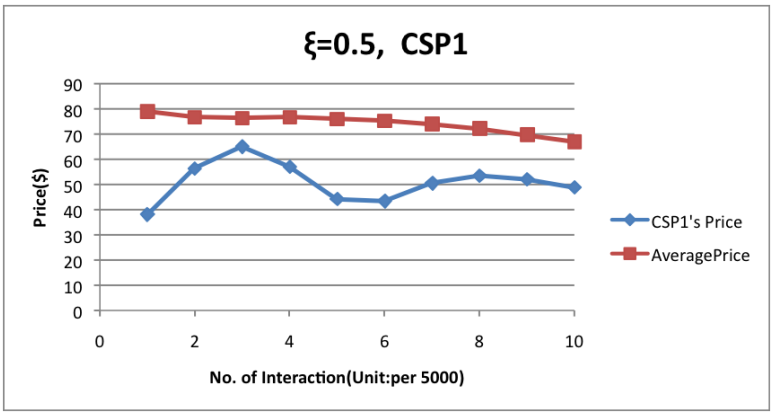


Figure 4.9: $\xi = 0.5$: Price adjustment based on both market price as well as price offered by same CSP in past iterations

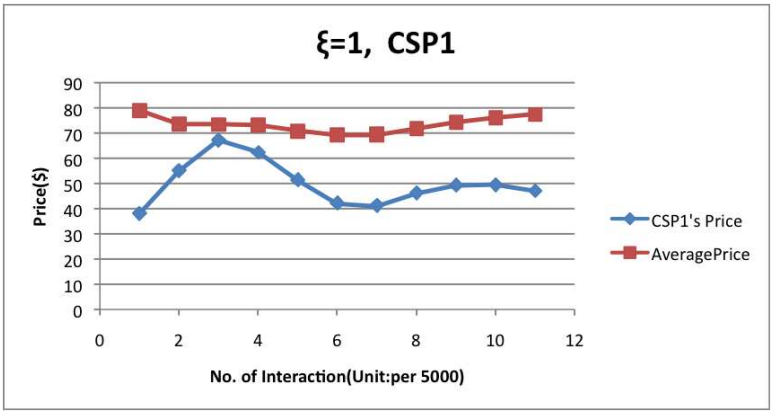


Figure 4.10: $\xi = 1$: Price adjustment based on only the price offered by same CSP in past iterations

4.3.6 Effect of Different settings of Expected Acceptance Rate

In this section we analyze the effect of expected acceptance rate on the revenue of CSPs for five different cases. Our simulation parameters are summarized in Tables 4.7 and 4.8.

- **Scenario 1:** All CSPs set their desired job acceptance rate $A_{thr} = 0.2$

Table 4.7: Simulation parameters for Section 4.3.6

Parameter	Values
Number of requests	50,000
Number of CSPs	10
Number of Users	1000
Minimum Price	50
Maximum Price	70

Table 4.8: Base Credit, Initial Price and Acceptance Rate for 10 CSPs

CSP id	Base Credit	Initial Price	Acceptance Rate				
			Scn1	Scn2	Scn3	Scn4	Scn5
1	0.764225	58	0.2	0.1	0.05	0.1	0.01
2	0.648288	64	0.2	0.1	0.05	0.1	0.05
3	0.778135	59	0.2	0.1	0.05	0.1	0.2
4	0.618787	64	0.2	0.1	0.05	0.15	0.2
5	0.639872	60	0.2	0.1	0.05	0.15	0.05
6	0.656864	64	0.2	0.1	0.05	0.15	0.15
7	0.768901	59	0.2	0.1	0.05	0.15	0.1
8	0.615769	62	0.2	0.1	0.05	0.2	0.1
9	0.606409	64	0.2	0.1	0.05	0.2	0.01
10	0.668191	59	0.2	0.1	0.05	0.2	0.15

(high) and the results are plotted in Figure 4.11. This is considered to be high because when there are 10 CSPs, on average each CSP will be selected with probability 0.1.

- **Scenario 2:** All CSPs set $A_{thr} = 0.1$ (normal) and the results are plotted in Figure 4.12.

- **Scenario 3:** All CSPs set $A_{thr} = 0.05$ (low) and the results are plotted in Figure 4.13.

- **Scenario 4:** CSPs have random values for A_{thr} from the set $\{0.1, 0.15, 0.2\}$ as summarized in Table 4.8 and the results are plotted in Figure 4.14.

- **Scenario 5:** CSPs have random values for A_{thr} as summarized in Table 4.8 and the results are plotted in Figure 4.15.

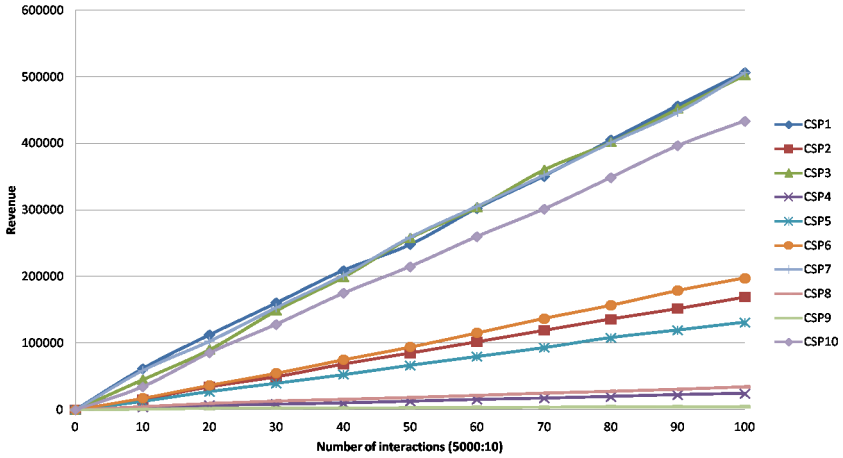


Figure 4.11: Analysis of revenue for acceptance rate $A_{thr} = 0.2$

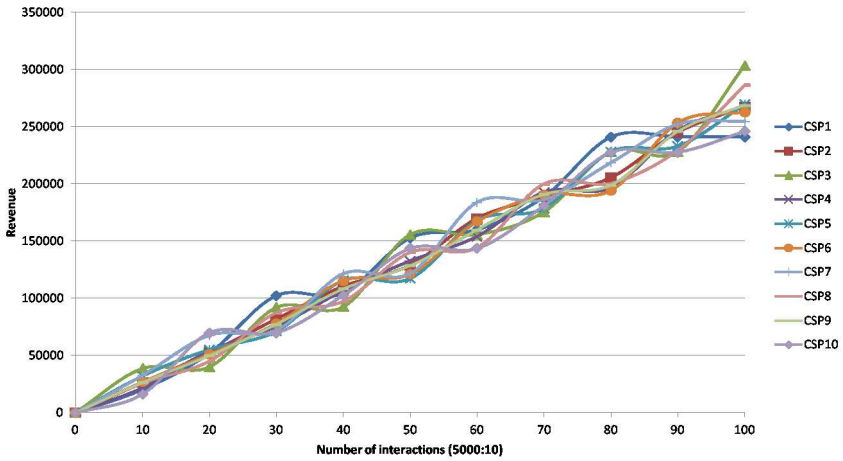


Figure 4.12: Analysis of revenue for acceptance rate $A_{thr} = 0.1$

When $A_{thr} = 0.2$, all CSPs set the expected acceptance rate to be high and hence the revenue obtained by the CSPs are based on the credit rating for the CSPs. For example, $CSP1$, $CSP3$ and $CSP7$ have the highest credit rating and hence get the highest revenue compared to other CSPs. Similarly, $CSP4$, $CSP8$ and $CSP9$ have the least credit rating and hence obtain a lower revenue compared to other CSPs. When $A_{thr} = 0.1$, all CSPs set their A_{thr} to be the normal expected

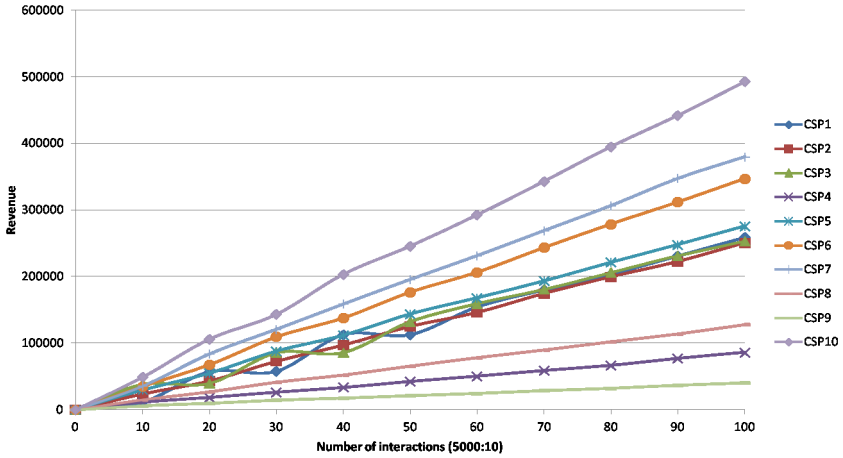


Figure 4.13: Analysis of revenue for acceptance rate $A_{thr} = 0.05$

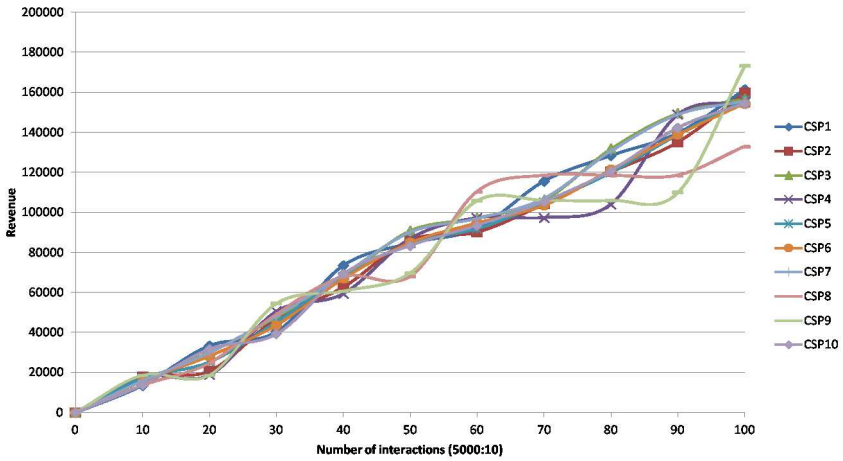


Figure 4.14: Analysis of revenue when acceptance rate A_{thr} is random; Scenario 4

acceptance rate and hence obtain fair revenue. When $A_{thr} = 0.05$, CSPs set their A_{thr} to be much lower compared to their average expected acceptance rate. Hence, the revenue obtained is unpredictable and we can not expect this case in a real market.

In reality, different CSPs may maintain different values for the expected accep-

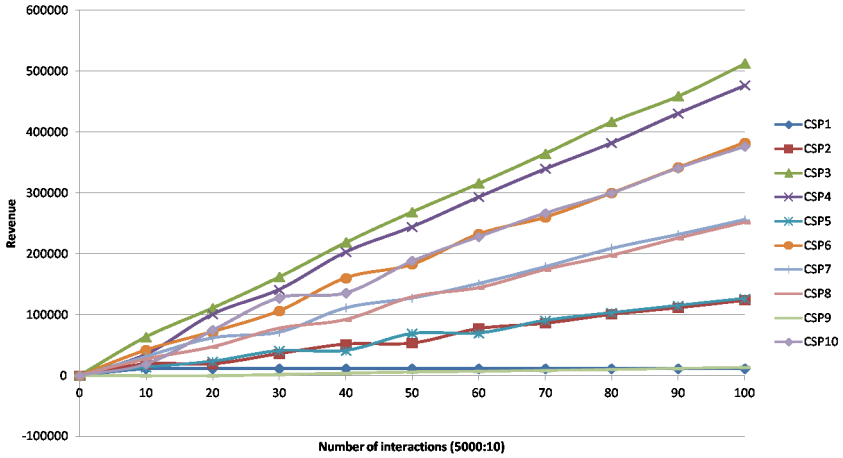


Figure 4.15: Analysis of revenue when acceptance rate A_{thr} is random; Scenario 5

tance rate. When A_{thr} is random, those CSPs with higher credit ratings show similar behavior whereas those CSPs (CSP 8 and CSP 9) with high A_{thr} and low credit rating show slight deviations in obtaining their revenue (Refer Figure 4.15. This results from the pricing strategies because when the credit is less and A_{thr} is high, those CSPs reduce the price offers to attract more users until they get the expected acceptance rate. We can also observe from Figure 4.15 that, for CSPs with same high A_{thr} ($CSP3$ and $CSP4$ in this example), the CSP with high credit ($CSP3$) gets higher revenue compared to the other CSP. Finally, CSPs with lower A_{thr} obtain very low revenue as expected.

4.3.7 Effect of the frequency in changing the Price offers

In this section, we analyze the effect of the frequency in changing the price offers on the revenue for different cases. In addition, we compare this values with the

Auction-based multiple Cloud Orchestration mechanism described in Chapter 3[1]. Values used for various simulation parameters are are summarized in Table 4.9.

Table 4.9: Simulation parameters for Section 4.3.7

Parameter	Values
Number of requests	1,00,000
Number of CSPs	5
Number of Users	1000
Minimum Price	45
Maximum Price	55
Plot value iteration	Every 10000

- **Scenario 1:** CSPs analyze the market conditions and change the price offer every $10,000$ iterations. The results are plotted in Figure 4.16.
- **Scenario 2:** CSPs analyze the market conditions and change the price offer every $1,000$ iterations. The results are plotted in Figure 4.17.

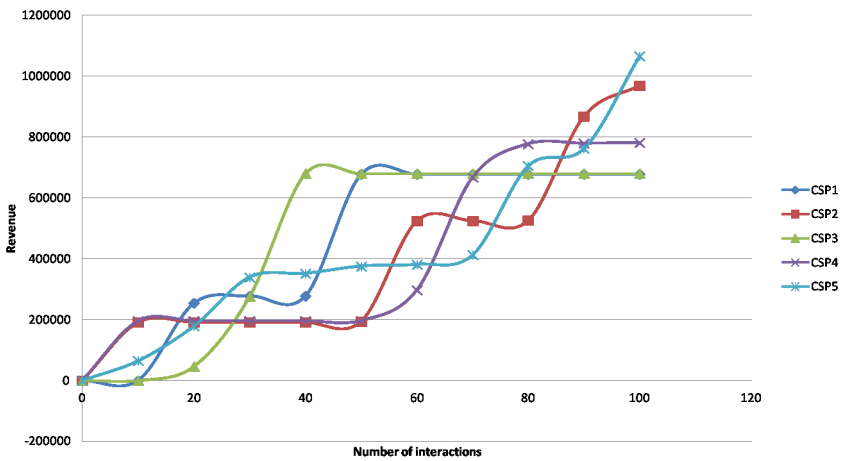


Figure 4.16: Effect of the frequency in changing the Price offers in revenue scenario 1

[1]Pricing mechanism in Incentive-based scheme described in Chapter 3 is based on individual user's affinity to different CSPs and hence we can not compare that in this case.

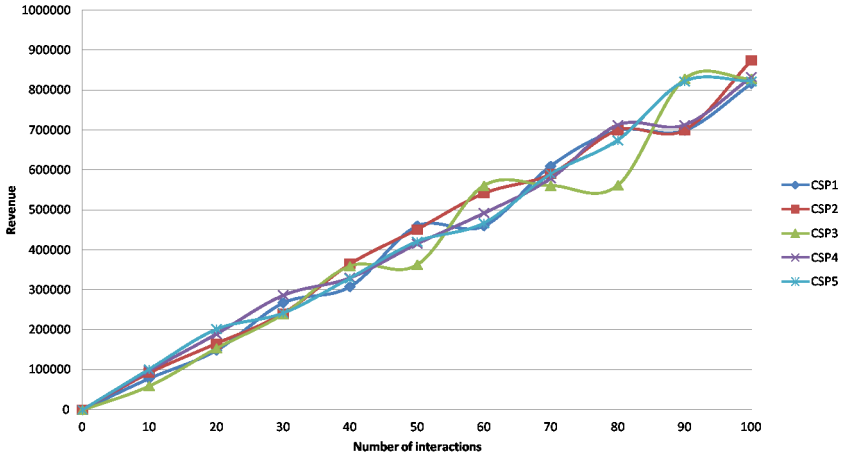


Figure 4.17: Effect of the frequency in changing the Price offers in revenue scenario 2

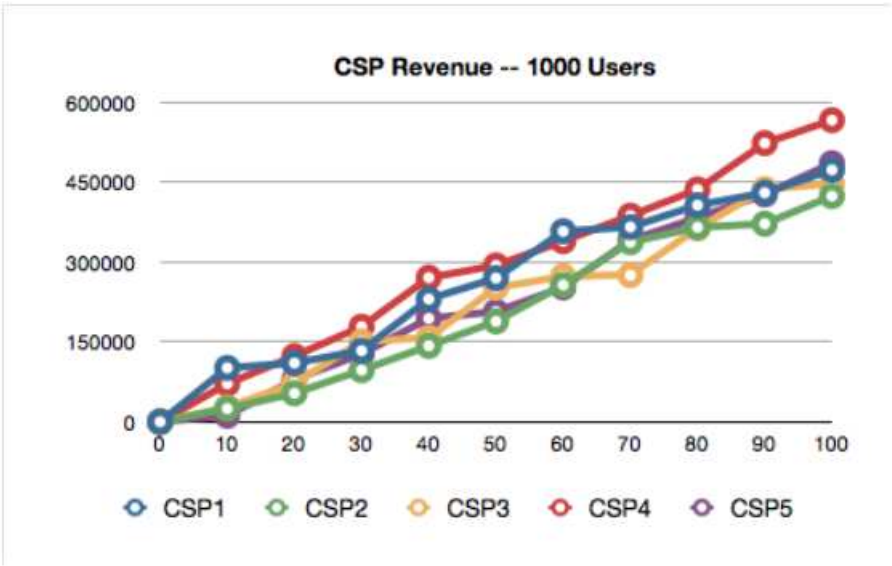


Figure 4.18: Revenue for auction based scheme proposed in Chapter 3

By comparing Figures 4.16 and 4.17, we can understand that when price offers are altered more frequently (Figure 4.17), revenue of different CSPs increase in a smooth manner and the revenue they generated are more fair compared to the case when price offers are altered less frequently (Figure 4.16). In real case, it

depends on the CSP's willingness. If the CSP is satisfied with its performance, it can slow its adjustment with the market and vice versa. An important conclusion from this study is that CSPs need not monitor and alter the price offer after every iteration. Even if they alter the price offers periodically, they can still adapt to the market conditions quickly.

In order to compare the performance of the present scheme with the Auction-based multiple Cloud Orchestration mechanism described in Chapter 3, we illustrate the relevant results in Figure 4.18. Compared to Figure 4.17, CSPs in the present model generate more revenue. During the auction, CSPs' revenues are more fairly distributed. However, the present model is more flexible. As discussed earlier, by changing CSPs' expected acceptance rate, our model can generate different revenues.

4.3.8 Comparison of different Broker arbitrage mechanisms

In this section, we conduct experiments to compare the performance of the present scheme to both auction based and incentive based schemes proposed in Chapter 3 using the simulation parameters described there. We also compare the results with the model without Broker. We conduct experiments for the present model with two cases one with lower budget requirements and the other one with higher budget requirements specified by the users. We analyze the revenue obtained as well as the Jain's fairness index and the results are plotted in Figures 4.19 and 4.20, respectively.

We can observe that the present scheme generates revenues which is comparable

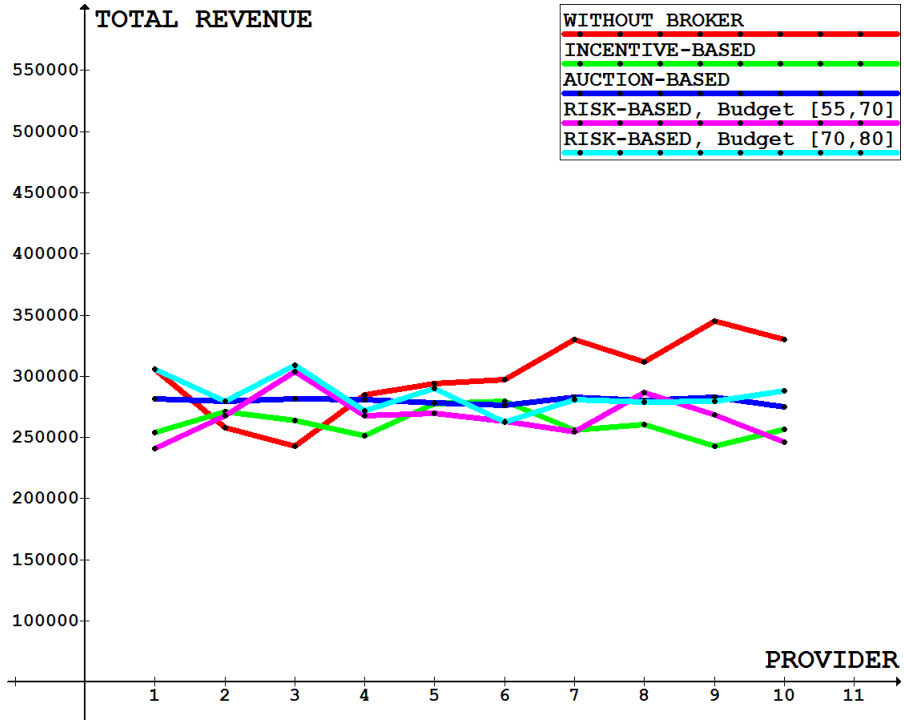


Figure 4.19: Comparison of revenue for various schemes

to the revenues generated by Auction-based and Incentive-based mechanisms for various budget requirements specified by the user. Further, we can also observe that, when the user has higher budget specifications, they are willing to pay more for the resources and hence the CSPs generate higher revenue. CSPs obtain this higher revenue because of the dynamic pricing strategies based on past transactions as well as the market price obtained from Broker.

We can also observe that our scheme has comparable values for Jain's fairness index (close to unity) with respect to the other two cases. Further, even if some of the CSPs generate lower revenue in the beginning, due to the market-based dynamic pricing strategies, they are able to get a fair revenue after some time.

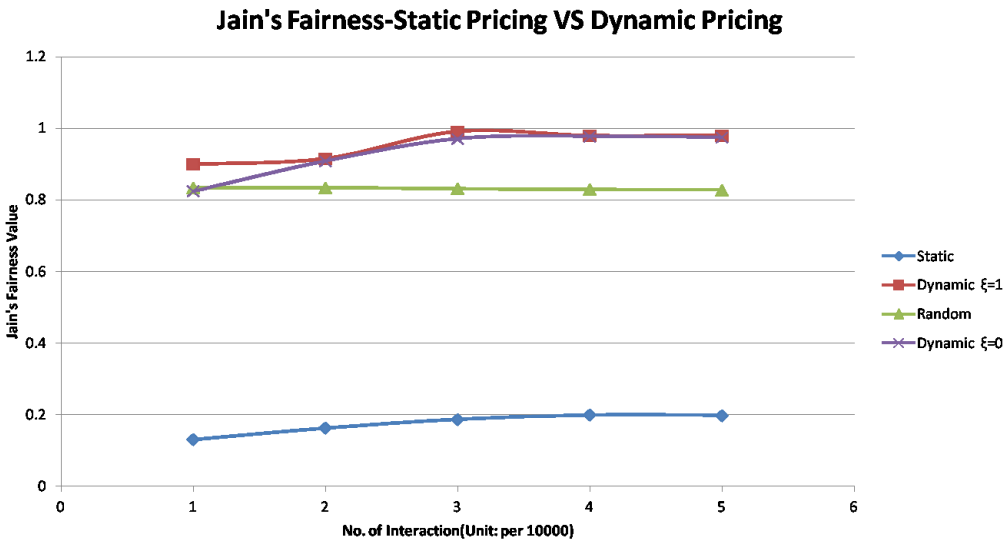


Figure 4.20: Comparison of Jain's fairness index for various schemes

4.3.9 Cloud market offering multiple services

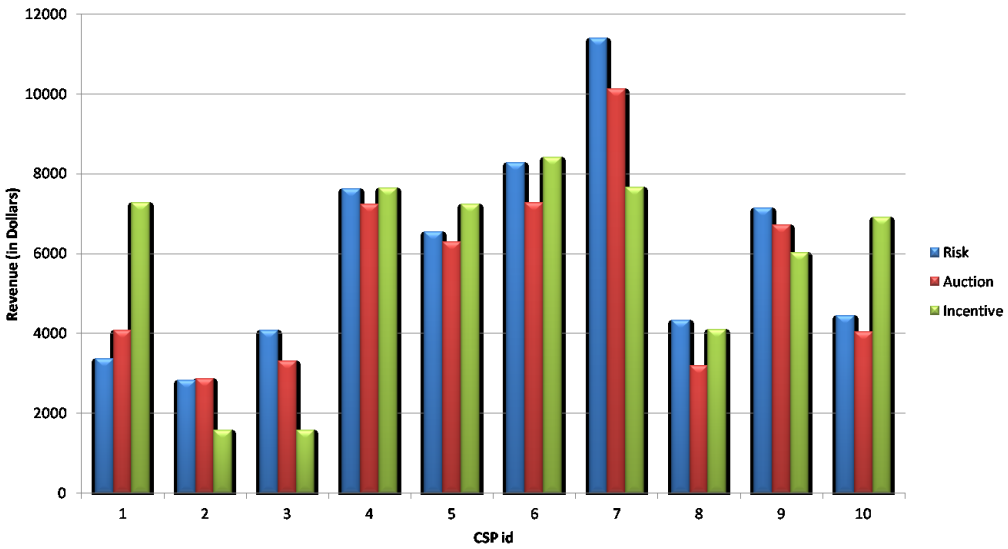


Figure 4.21: Revenue obtained when CSPs offer different products

In order to demonstrate that our scheme can seamlessly incorporate CSPs offering multiple resources and users having diverse resource requirements, we

consider CSPs offering three different services, Compute, Storage and Data for the experiments as described in Section 3.6.5 of Chapter 3. We conduct experiments in this section for 1000 iterations with 100 users and the results obtained are plotted in Figure 4.21. We can observe that, CSPs offering multiple services obtain more revenue compared to CSPs offering a single service. This is because they get more opportunities based on user requests. Further, it is interesting to note that CSPs offering the same types of resources are treated fairly by our scheme. For example, CSP 5 and 9 (offering Storage and Data) get similar revenues. This analysis reveals that our scheme can seamlessly handle CSPs offering multiple types of services and users with diverse resource requirements.

4.4 Chapter Summary

In this chapter, we proposed a risk-aware Cloud Broker Arbitrage mechanism to connect various CSPs and users through a Broker-mediated system. We proposed a mathematical model for the utility function for the users to choose appropriate CSP from time to time and considered various user specific parameters such as user's risk-bias, trust and reputation of CSP, price offered by the CSP and user's budget requirements. Further, we formulated a function for the CSPs to make dynamic price offers based on the last transactions, CSP's expected and real acceptance rates as well as the current market price.

Later we conducted extensive simulation studies and analyzed the effect of various factors such as the effect of dynamic credit mechanism used in our model, various dynamic pricing strategies that are supported by our system and the effect of different system parameters in the CSP's revenue. Finally, we also compared our

model with other models and showed that our scheme is effective under various cases.

PART II: CLOUD AGGREGATION MECHANISMS

Chapter 5

Cooperative Game-theoretic Approaches for Cloud Aggregation

5.1 Introduction

In the first part of this thesis, we have described an architecture for a Cloud Broker and discussed several Cloud Broker Arbitrage mechanisms to provide opportunistic choices and competition among different CSPs. In the following chapters, we describe two Cloud Broker Aggregation mechanisms for deploying user applications among multiple resources to meet certain customer requirements based on the Broker architecture described in Chapter 3. We devise strategies for deploying compute and data intensive applications such as Bag-of-Tasks (BoT) applications and divisible load applications on IaaS Compute Cloud environments based on user requirements. Note that Broker may be used for task aggregation within

one Cloud or across multiple Clouds based on the requirements imposed by the user. For example, an organization may have a private Cloud within which they may want to deploy the tasks with the help of a Broker. The Broker can also aggregate tasks among multiple CSPs offering same or different types of resources to meet some user objectives.

In this Chapter, we consider task aggregation in a private Compute Cloud (which is equally applicable for aggregation among multiple CSPs having same types of resources) running on a particular platform in which all computing resources have identical characteristics. (eg: standard instances of Amazon EC2 running on IBM Informix dynamic servers). Based on the requests received for the resource instances, Broker needs to employ efficient strategies to allocate optimal number of resources to achieve certain goal such as minimizing task execution time or maximizing revenue, which is beneficial to both CSPs and users. In our model, users submit few parameters pertaining to the tasks such as task deadline or budget requirements and the Broker calculates the optimal resource assignment.

The strategies described in this Chapter are suitable for scheduling both independent tasks as well as workflow tasks. We employ bargaining approaches [68] which are propounded in the literature [69] for devising our task aggregation strategies. A price based non-cooperative bargaining theoretic model for a mobile grid environment, has been discussed in [70]. A utility-based resource negotiation approach for resource management in grid based content distribution network is proposed in [71]. In [72], noncooperative alternating-offers bargaining game is used for formulating a pricing strategy in distributed computing systems. All these proposals emphasize on pricing and they require a few iterations to reach

the equilibrium.

In [73], the authors use Nash bargaining solutions for job allocation in a grid environment. They assume that the grid providers can not handle the requests themselves and hence different providers cooperate to guarantee Quality of Service (QoS). For a Cloud environment, this may not be true. Further, the above models are not capable of handling real-time task arrivals and fluctuations in user requirements with time. A CSP has a huge free pool of resources available for customers and it requires strategies to efficiently allocate the available resources for tasks arriving to the Cloud. We consider this situation in our framework.

5.2 Cooperative Game-Theory Framework

We assume that the tasks to be executed in the Compute Cloud are known *a priori*. This is a reasonable assumption for a Cloud because tasks to be executed in Cloud are generally submitted well in advance and they usually have large processing time (of the order of hours or days) [74]. Further, tasks within certain applications such as compute-intensive workflow applications are known in advance. However, they may not know the exact amount of resources needed until the run-time. Later we show that our models can handle task dynamics as well as real-time task arrivals up to a great extent. We consider computation-intensive tasks that demand many Virtual CPU Instances (VCIs) and need to meet certain requirements such as deadline and/or budget. Further, without any loss of generality, we assume that the scheduling details such as the task distribution values for different CSPs (if the aggregation is performed on multiple CSPs), the individual billing information etc are abstracted from the User and the Broker

takes care of these information management and Broker maintains the information such as integrated billing.

Consider there are N tasks $(t_1, t_2, ...t_N)$ present in the system waiting for execution. The players or bargainers in this problem are these tasks. Task specifies two parameters, $\{AET^i, t_d^i\}$, average execution time and deadline for the task when deadline[1] is the primary criteria and budget requirements when budget is the primary criteria. The complete model of the system, based on the Broker architecture described in Chapter 3, is illustrated in Figure 5.1. Note that in the

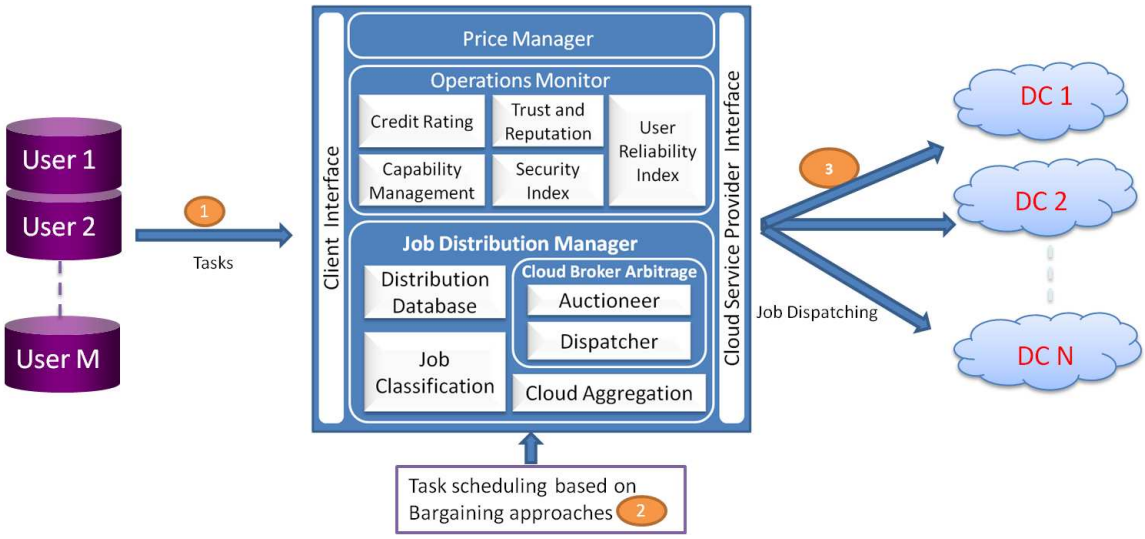


Figure 5.1: Architecture for the proposed bargaining model. Here, DC stands for Datacenter and these datacenters may belong to one or more CSPs

proposed aggregation mechanisms, we focus on task scheduling to satisfy user specifications and ignore other parameters such as handling reliability and trust. It can be observed that such parameters can be specified by the users when they submit the tasks and Broker can consider a subset of CSPs satisfying such criteria for the task scheduling in a seamless manner.

[1]Deadline is the time on or before which the task should be completed.

Let $R = \{R_1, R_2, ...R_N\}$ be the bargaining domain or the feasible set of all possible outcomes (VCIs) required for each task. R is assumed to be convex, closed and bounded sets of $\Re^N$. Each player i has a *disagreement point* d_i which is the minimum number of VCIs required to complete the execution based on certain requirements. The disagreement point for the game can be represented as $d = (d_1, d_2, ..d_N)$. The pair (R, d) is the bargaining problem under consideration. Below we present the required background material on the complete formulation of optimal solutions for NBS and RBS based strategies.

5.2.1 Nash Bargaining Solution (NBS)

The Nash Bargaining Solution (NBS) allows us to assign fair amount of resources for various tasks present in the system [68]. Let $r = f(R, d)$ be an NBS which satisfies the following four axioms [68]:

- A1. *Pareto Optimality*: If (R, d) is a bargaining problem with $r, r' \in R$ and $r'_j > r_j, \forall j$, then $f(R, d) \neq r$.
- A2. *Independence of Linear Transformations*: Let $y(.)$ be any positive affine linear transformation, $y(r) = f(y(R), y(d))$.
- A3. *Symmetry*: If bargaining problem is symmetric, for any two users k and m where $d_k = d_m$, then $f_k(R, d) = f_m(R, d)$.
- A4. *Independence of Irrelevant Alternatives*: For every $r' \in R'$ where $r' = f(R', d)$, if $R \subseteq R'$, then $f(R, d) = r'$.

Below we attempt to explain the physical significance of the above axioms. The linearity axiom $A2$ says that, the solution is not affected if the performance

objectives are affinely scaled. Imposing the axiom of symmetry $A3$ assumes that all players have equal bargaining skills. But in reality, the bargaining may be influenced by other parameters such as the strategies employed by the players for bargaining. This can be termed as the bargaining power α_i for a player i. Such bargaining games are called as asymmetric bargaining games [68]. The axiom $A4$ tells that the bargaining point is not affected by enlarging the domain. For the asymmetric NBS model [68], we can define the utility function for the task i as $R_i^* = (R_i - d_i)^{\alpha_i}$. where R_i^* is the optimal number of resources derived based on NBS for task i. Now, $R^* = \{R_1^*, R_2^*, ..., R_N^*\}$ which satisfies the axioms of generalized NBS are the optimal set of resources for the tasks. The optimization problem can be defined as $(P1)$:

$$R^* = \arg\max_{R} \prod_{i=1}^{N} (R_i - d_i)^{\alpha_i} \tag{5.1}$$

subject to

$$(R_1, ..., R_N) \in R, \tag{5.2}$$

$$R_i \geq d_i \ \forall i, \quad \text{and} \tag{5.3}$$

$$\textstyle\sum_{i=1}^{N} R_i \leq R_{tot} \tag{5.4}$$

Here R_{tot} is the total amount of resources available. The objective function in Equation (5.1) captures the proportional fairness. This means that every task will get resources in proportion to what it has requested. Due to the nature of Cloud, i.e. number of resources available is far more than the requests, we are trying to allocate more resources to users than the required number of resources.

This will help to finish the task before deadline and free the resources earlier. The constraint $R_i \geq d_i \ \forall i$ says that each task i should get at least the minimum number of resources d_i required to finish the task before deadline. Similarly the constraint $\sum_{i=1}^{N} R_i \leq R_{tot}$ specifies that the total number of allocated tasks can not exceed the total number of resources available. To obtain feasible bargaining outcomes, $\sum_{i=1}^{N} R_i \leq R_{tot}$ must be satisfied. Assuming $\sum_{i=1}^{N} R_i = R_{tot}$, the NBS for user i is given as[1]

$$R_i^* = \frac{\alpha_i (R_{tot} - d_{tot})}{\alpha_{tot}} + d_i \tag{5.5}$$

where α_{tot} is the sum of all bargaining powers of all tasks and d_{tot} is the sum of the minimum number of resources required for all tasks. The solution offered by NBS is useful and directly applicable when the set of tasks are known in advance and the Broker needs to allocate the resources for these tasks in a fair manner. Moreover, it makes use of all available free resources for allocation which leads to high resource utilization. In short, NBS results in fair resource allocation which maximizes the free resource utilization.

On the other hand, although NBS offers an attractive solution, it may be noted that it only takes care of individual's gain and does not care about how much others have given up. In order to take this fact into consideration, we adopt another solution, which is an extension of NBS as described below.

5.2.2 Raiffa-Kalai-Smorodinsky Bargaining Solution (RBS)

In order to ensure that one's gain should be proportional to its maximum gain (in other words, every player should give same weight for individual gain and other

[1]Readers may refer to the proof in [68].

player's losses), Kalai and Smorodinsky, and Raiffa [68] proposed a new solution. They retained axioms $A1 - A3$ in Nash's solution and added a new axiom $A4'$ as follows:

- $A4'$: *Monotonicity*: For any $r' \in R'$ where $r' = f(R', d)$, if $R \subseteq R'$ and $\sum_{n=1}^{T} r'_{k,n} \geq \sum_{n=1}^{T} r'_{k,n}$, then $f_k(R', d) \geq f_k(R, d)$.

Consider the same game we defined before with T players. We can define [75] the ith player's preference function with minimum utility d_i and maximum utility R_i^{max} as follows:

$$v_i(\beta) = [(R_i - d_i) + \frac{\beta}{N-1}(\sum_{j \neq i}^{N} R_j^{max} - R_j)]^{\alpha_i} \tag{5.6}$$

where, β is the weighing factor to measure the trade-off between one's gain and another's loss. When these two factors are given equal weightage, i.e. substitute $\beta = 1$ in equation (5.6), we obtain RBS. So we can write the utility function for RBS as follows:

$$v_i(\beta) = [(R_i - d_i) + \frac{1}{N-1}(\sum_{j \neq i}^{N} R_j^{max} - R_j)]^{\alpha_i} \tag{5.7}$$

Given this utility function, we can write the RBS optimization problem $(P2)$ as follows:

$$\tilde{R}^* = \arg\max_{R} \prod_{i=1}^{N} [(R_i - d_i) + \frac{1}{N-1}(\sum_{j \neq i}^{N} R_j^{max} - R_j)]^{\alpha_i} \tag{5.8}$$

subject to

$$(R_1, ..., R_N) \in R, \tag{5.9}$$

$$R_i \geq d_i \ \forall i, \tag{5.10}$$

$$R_i \leq R_i^{max} \ \forall i, \quad \text{and} \tag{5.11}$$

$$\sum_{i=1}^{N} R_i \leq R_{tot} \tag{5.12}$$

where R_i^{max} is the maximum resources that can be allocated for task i. To obtain feasible bargaining outcomes, $\sum_{i=1}^{N} R_i \leq R_{tot}$ must be satisfied. Assuming $\sum_{i=1}^{N} R_i = R_{tot}$, the RBS for user i is given as[1] $\tilde{R}_i^* = min\{\tilde{R}_i^*, R_i^{max}\}$ where $\tilde{R}_i^*$ can be expressed as shown in Equation (5.13).

$$\tilde{R}_i^* = \alpha_i^n R_{tot} + \frac{(1 - N\alpha_i^n)R_{tot} + (N-1)(d_i - \alpha_i^n d_{tot}) + R_i^{max} + ((N-1)\alpha_i^n - 1)\sum_{i=1}^{N} R_i^{max}}{N} \tag{5.13}$$

In Equation (5.13), $\alpha_i^n = \frac{\alpha_i}{\sum_{i=1}^{T} \alpha_i}$ is the normalized bargaining power with $\sum_{i=1}^{N} \alpha_i^n = 1$ and d_{tot} is the sum of the minimum resources required for all tasks. $\tilde{R}_i^*$ is the optimal resources derived based on RBS for task i.

The Nash and Raiffa solutions that we derived satisfy the axioms, and we now show that the results are on Pareto optimal boundary. When Nash solution maximizes the product of the gain of all players, the Raiffa solution considers how much other players gave up, in addition to one's gain. We now demonstrate

[1]Readers may refer to the proof in [68].

the workings and solutions of the above proposed strategies using an illustrative example with two tasks.

Example 1. Suppose there are 100 virtual CPU instances available, i.e. $R_{tot} =$

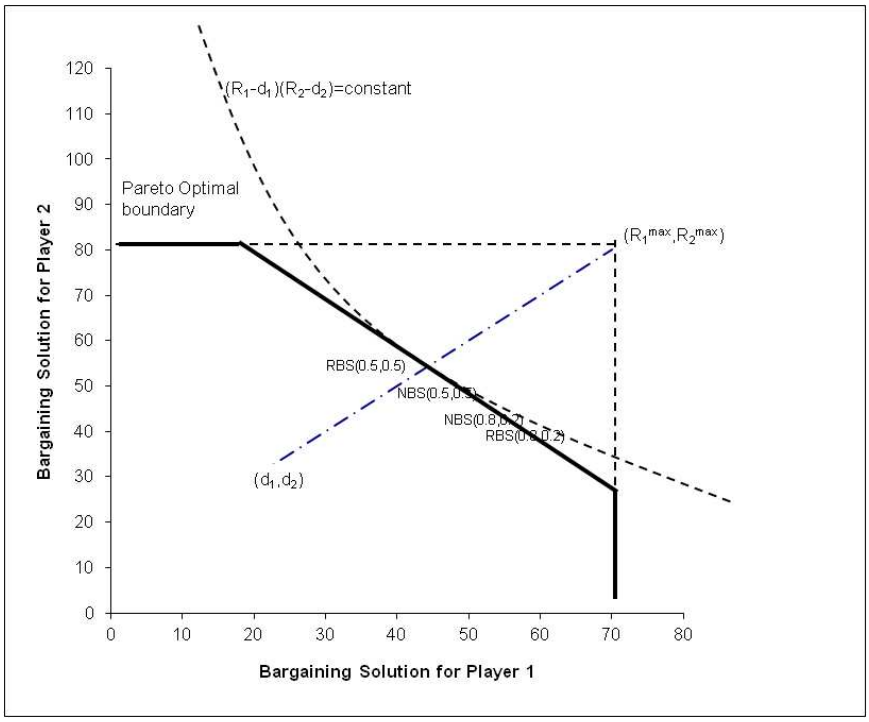


Figure 5.2: Geometrical Interpretation of Nash and Raiffa solutions

100 and the minimum and maximum requirements for two tasks are $d = (20, 30)$ and $R^{max} = (70, 80)$ respectively. Consider a case with equal bargaining powers (symmetric) for both tasks. i.e. $\alpha = (0.5, 0.5)$. Using equations (5.5) and (5.13), we obtain the solution as $R^* = (45, 55)$ and $\tilde{R}^* = (43, 57)$, for the respective cases.

Consider another case of asymmetric bargaining power with $\alpha = (0.8, 0.2)$. This means that player 1 increases his bid. The solution is $R^* = (60, 40)$ and $\tilde{R}^* = (61, 39)$. A geometrical interpretation of this solutions are described in Figure

5.2. The solid line is the Pareto boundary. From the geometrical interpretation as described in [75], the symmetric Nash solution lies on the tangent to the hyperbola $(R_1 - d_1)(R_2 - d_2) = constant$ and the symmetric Raiffa solution is the intersection point between a line from (d_1, d_2) to (R_1^{max}, R_2^{max}) and the Pareto optimal boundary. For the asymmetric case, player 1 has more bargaining power and hence got more resources in both cases.

We can make the following observations from this analysis. In both NBS and RBS, the allocation depends on the bargaining power. So even if two tasks have same value for disagreement point, depending upon the bargaining power, they may get a different amount of resources for their task execution. Further, NBS is a special case of RBS and we can obtain it by substituting $\beta = 0$ in equation (5.6). In this case, the players do not care about other player's loss. Whereas, we can observe that RBS takes into account both disagreement point and the maximum resource requirements to obtain the optimal allocation of resources. Further, if a CSP offers multiple kinds of resources, it can use this algorithm on each resource type separately to arrive at the optimal solution.

5.3 Performance Evaluation and Discussions

In this section, we perform rigorous simulation experiments to demonstrate how the above presented approaches are applied in handling the resource allocation problem in a variety of situations. To demonstrate this, we consider a general case with 300 tasks and we derive the bargaining power in three different ways:

- ***Deadline:*** Bargaining power is derived based on deadline. This means

critical tasks with short deadlines have higher bargaining power and tasks with longer deadline have lower bargaining power.

- ***Budget Requirements:*** Bargaining power is derived based on the budget constraints proposed by the tasks. This means, tasks which can afford to pay more, will get more resources still satisfying the disagreement point.

- ***Deadline and Budget:*** Bargaining power is derived based on both deadline requirements as well as the budget requirements. This means short deadline tasks which can afford to pay more, will get more resources.

5.3.1 Resource allocation based on Deadline

We derive the bargaining power based on the deadline as follows. AET^i values are randomly chosen within the range $[51, 100]$ and t_d^i values are within the range $[10, 50]$ for any task i. Then bargaining power α_i can be calculated as $100 - t_d^i$ for any task i. This α_i for all tasks are normalized such that it lies between 0 and 1. Then we consider a Compute Cloud with 3000 virtual instances of CPUs available for allocation. We obtain the disagreement point for a task i as $d_i = \frac{AET^i}{t_d^i}\ \forall i$. For simplicity, we calculate R_i^{max} as $R_i^{max} = 2 * d_i$ for RBS. In practical situations, the Brokers can obtain user requirements and make use of different approaches to derive d_i and R_i^{max}.

Figure 5.3 plots the percentage of available resources allocated in both schemes. In this graph, we can observe that if we allocate only based on d_i, there are many resources not being allocated. In this example, only around 40% of the total resources are used. We can observe that NBS efficiently allocates more than

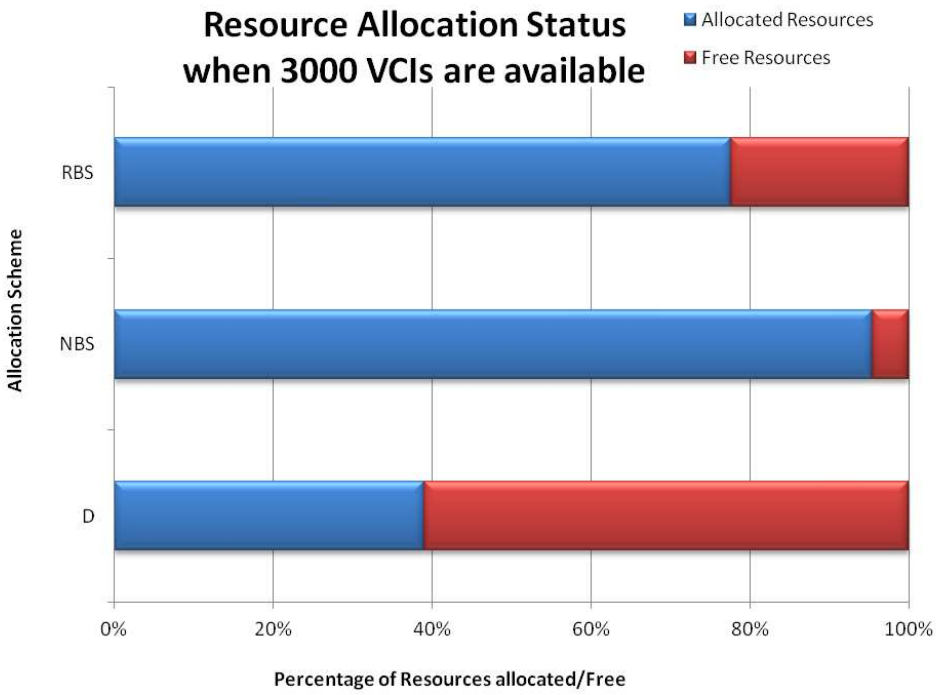


Figure 5.3: Percentage of Resources allocated/Free with $R_{tot} = 3000$ and $T = 300$

95% of the resources and hence resource utilization is high. On the other hand, for RBS, we note that approximately 78% of resources are allocated. Further, it allocates resources more evenly considering the maximum resource requirements and hence it is more efficient to finish the overall task execution in shorter time, especially for longer deadline tasks.

Handling Auto-Elastic property of Cloud

For various tasks in compute-intensive applications (such as workflow applications), especially for data analytic applications, the resource demand varies with time. Cloud environments have abundant computational resources and resources are assigned as and when demand exists which is referred to as *auto-elasticity*. To

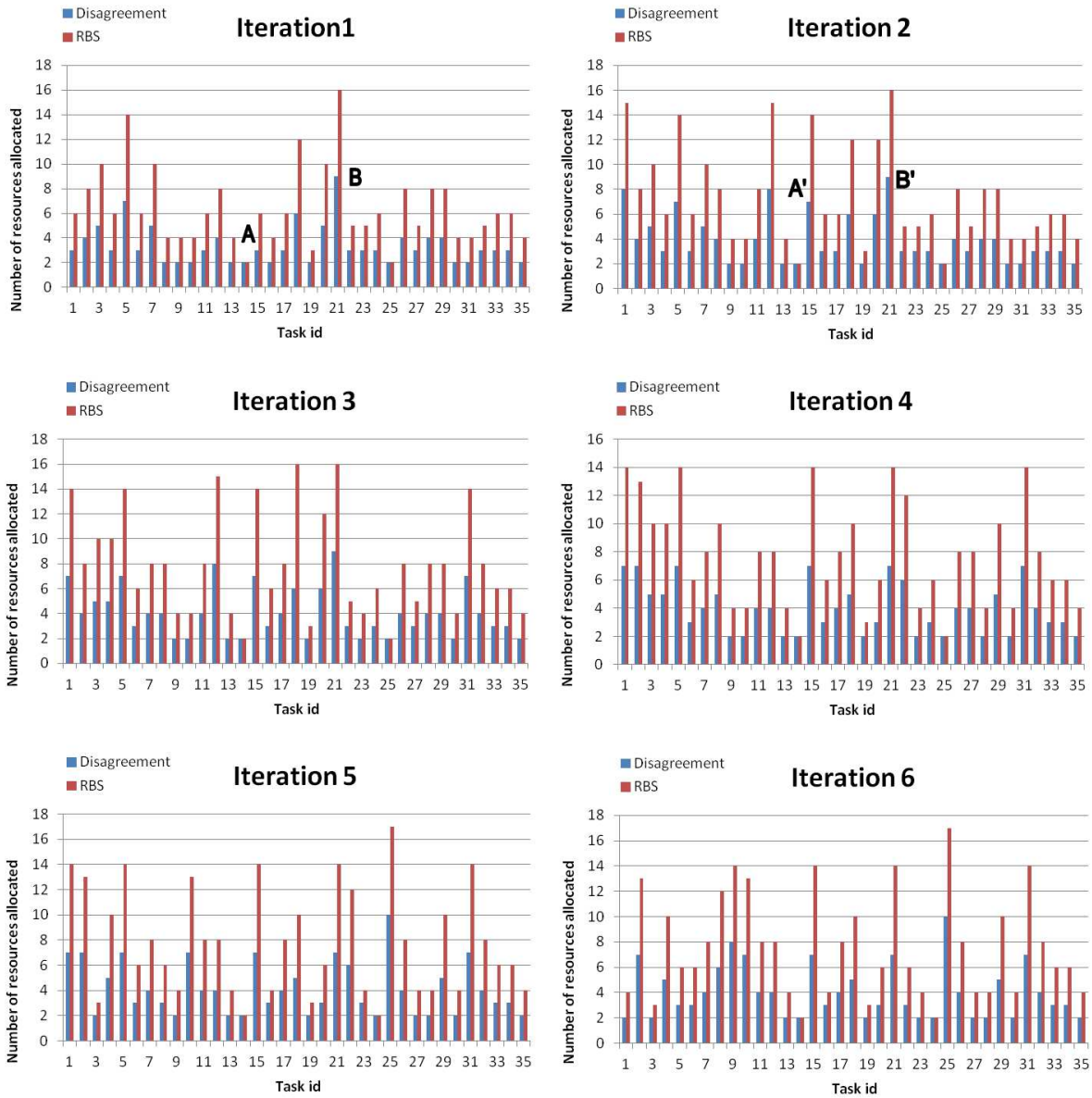


Figure 5.4: Number of resources allocated in 6 iterations with dynamic change in demand

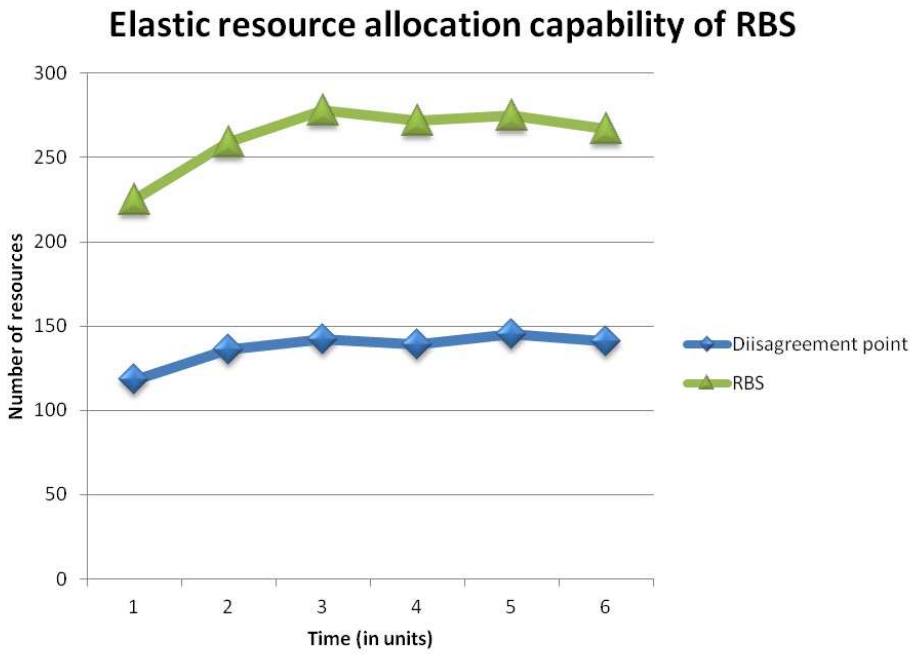


Figure 5.5: Auto-elasticity of Cloud when the demand varies with time with $R_{tot} = 300$ and $T = 35$

understand the efficiency of RBS in handling this property, we have conducted a set of experiments wherein the requirements of tasks vary with time. In the first set of experiments, we considered $R_{tot} = 300$ and $T = 35$. We have varied the demand of at most $k = 5$ randomly chosen tasks in each iteration. Figure 5.4 shows the individual resource allocation for 35 jobs in 6 different iterations. A summary of the total resources allocated in these 6 iterations are plotted in Figure 5.5. From these figures, we can observe that, RBS can handle variations in resource requirements efficiently. In these experiments, resources are allocated to tasks having more demand. Meanwhile, the algorithm does not change any other task's existing allocations even though the new calculation may demand a resource reallocation for these tasks (for example, refer to tasks A and B pointed

in Figure 5.4). This means that without changing any existing allocations, RBS is able to take care of the auto-elastic property of Cloud up to a great extent.

To understand this effect on NBS and RBS in a large scale situation, we have

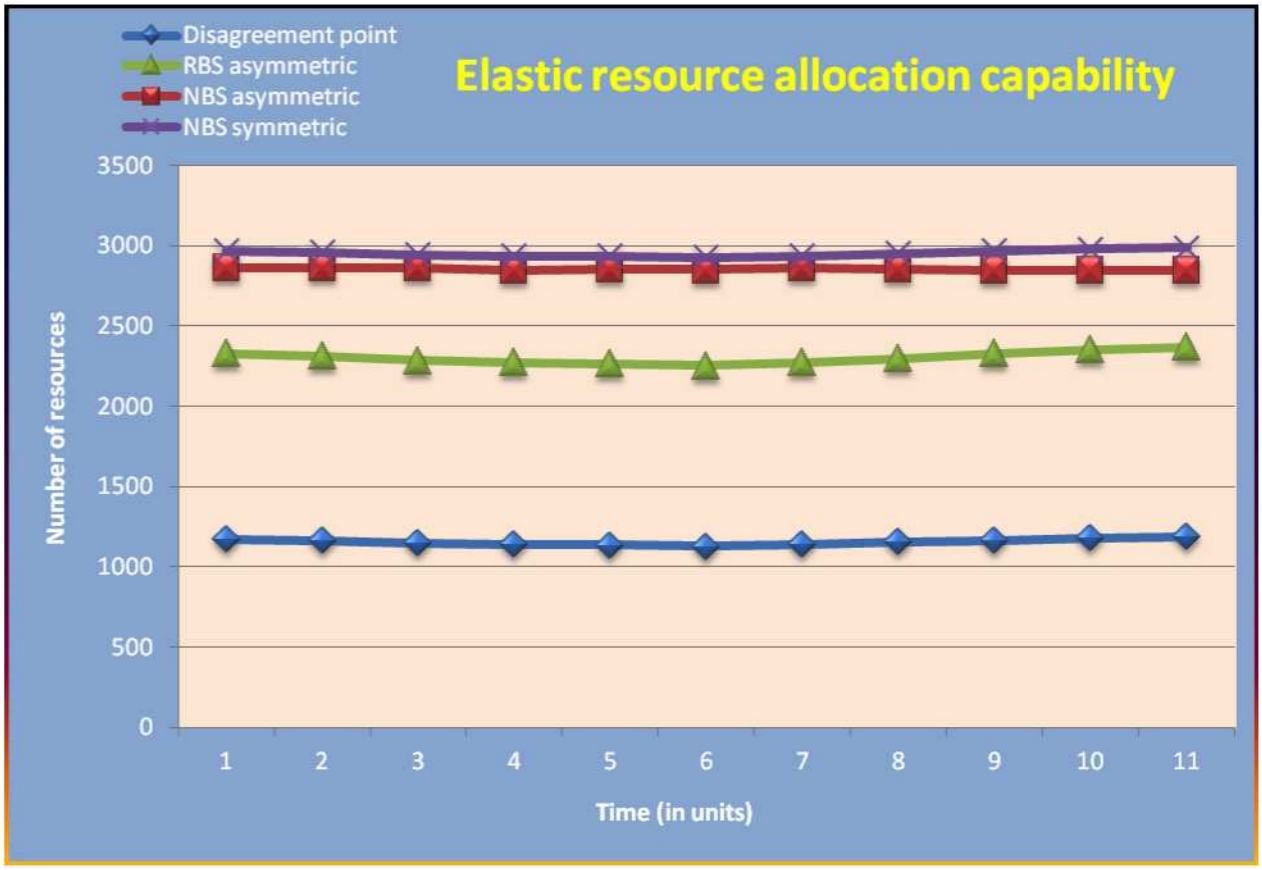


Figure 5.6: Auto-elasticity of Cloud when the demand varies with time with $R_{tot} = 3000$ and $T = 300$

repeated this experiments with $R_{tot} = 3000$ and $T = 300$. Here up to 50 randomly selected jobs change their demand in every iteration and the results are plotted in Figure 5.6. As we can observe from this figure, NBS allocate maximum number of resources to the existing tasks and hence it may need to re-allocate when the task demand changes with time. Among them, asymmetric NBS per-

forms slightly better than symmetric NBS due to the influence of α in making the optimal allocation. Whereas, as RBS consider maximum resource requirements, it can efficiently handle the auto-elastic property without affecting the existing allocations. So we can see that when the task requirements are stable, NBS is a better choice and when the task requirements change dynamically, then RBS could be a better choice. For a set of tasks with mixed task requirements, Broker can choose an appropriate value for β in Equation (5.6) in order to arrive at the optimal allocation.

Effect of real-time task arrival on RBS

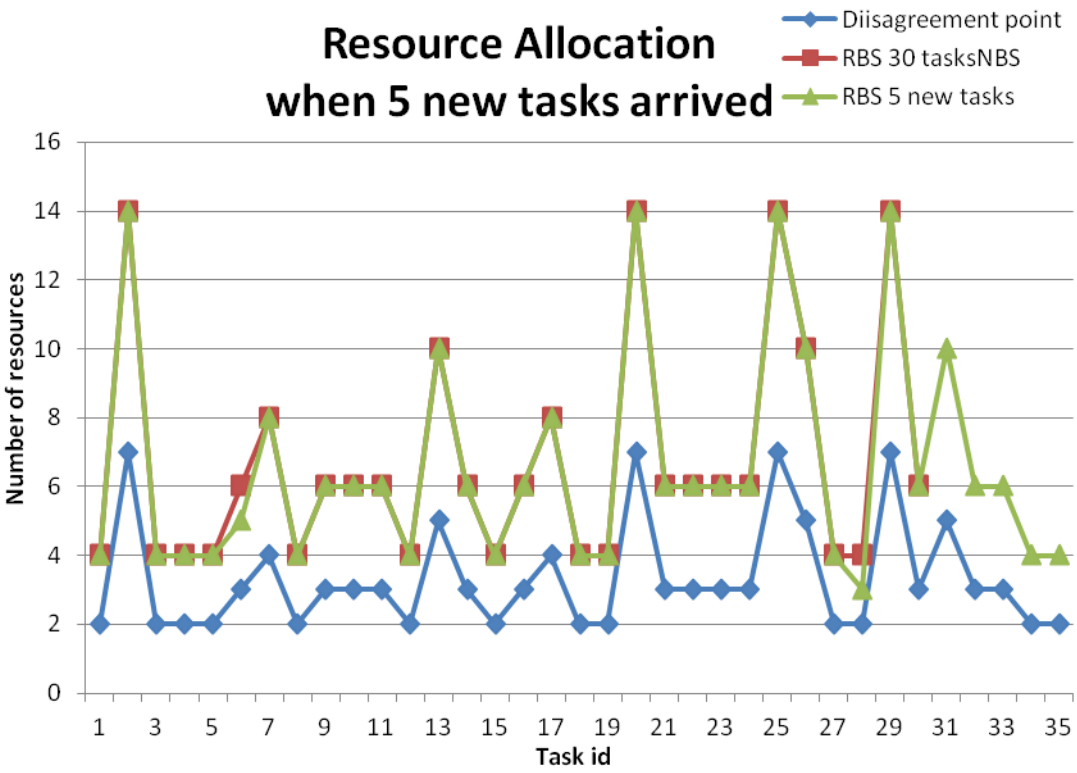


Figure 5.7: Resource allocation on RBS in two cases

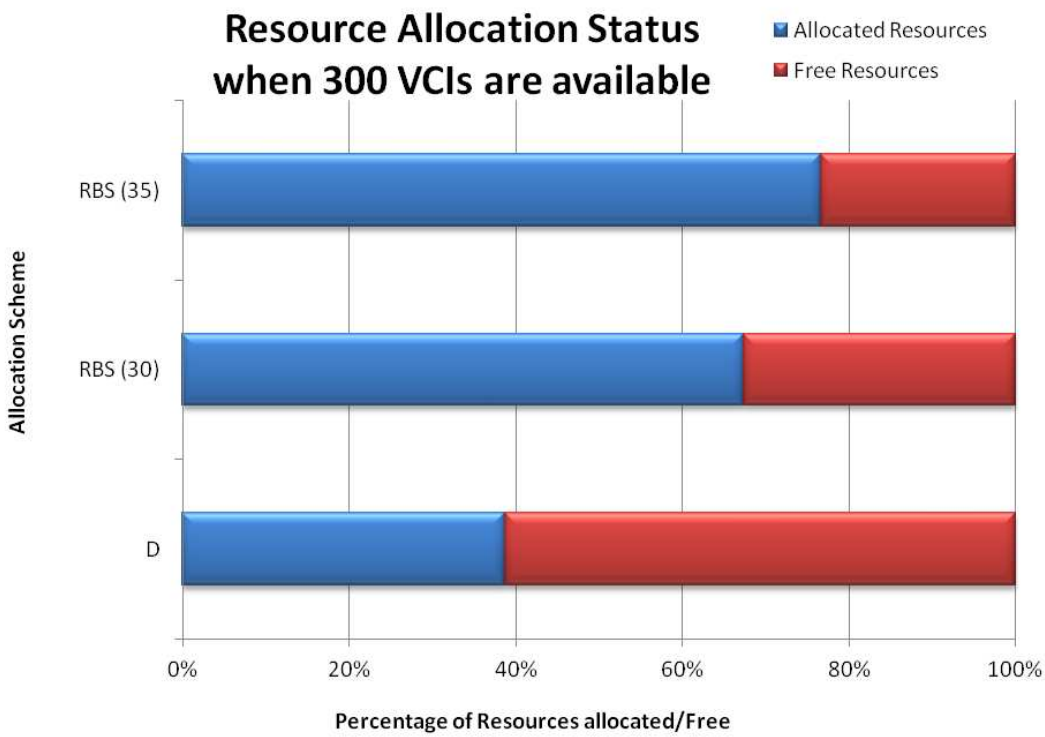


Figure 5.8: Percentage of Resources allocated/Free on RBS in two cases with $R_{tot} = 300$ and $T = 30$

In order to demonstrate the real-time task handling capability of RBS, we consider 30 tasks and the resource allocator derives the optimal resource allocation using RBS. Now consider 5 new tasks arriving at the Cloud. The aggregation unit in the Broker's Job Distribution Manager (JDM) run the RBS algorithm and derives the optimal number of resources for all the 35 tasks. The minimum number of resources required and the optimal number of resources derived are plotted in Figure 5.7. Figure 5.8 shows the percentage of resources allocated and freed in both cases. From Figure 5.8, we note that 65% (approx. 200 out of 300) of resources were used for allocation in first case. Later when 5 new tasks arrive, we observe that RBS consumes 77% resources (approx. 230 resources) and hence

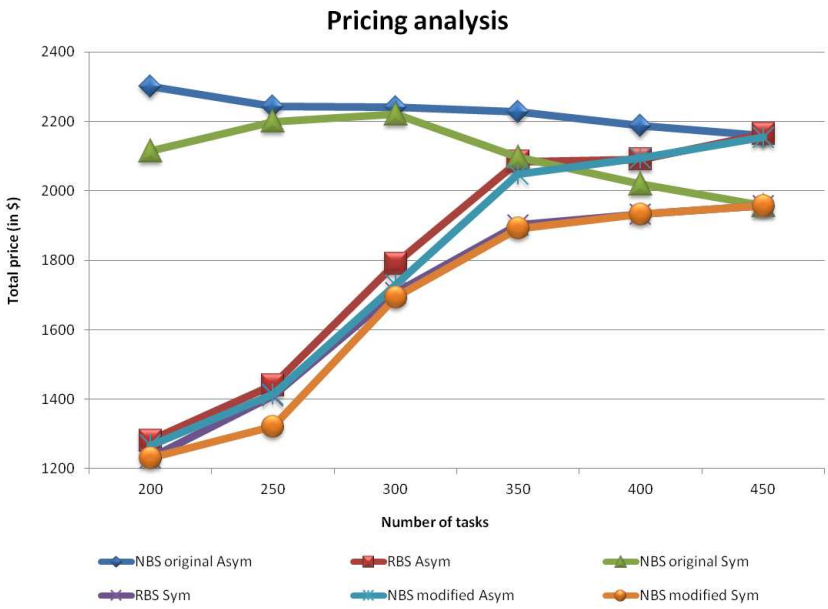


Figure 5.9: Analysis of pricing effects with change in number of tasks present

the allocator can allocate resources for these new tasks without affecting any of the existing allocation. Thus RBS has the ability to handle real-time task arrivals in a Cloud environment.

5.3.2 Budget requirements based resource allocation: Asymmetric pricing schemes

In a Cloud environment, a CSP is most often interested in a pricing analysis. In this section, we conduct experiments for analyzing the pricing aspects for both symmetric (equal bargaining power for all tasks in the system) and asymmetric (Users can specify how much they can afford to pay for the resources) cases of NBS and RBS. In case of asymmetric schemes, we allow the tasks to specify a price per resource in a range $[K_{min}, K_{max}]$ with $K_{min} = 50$ cents and $K_{max} = 1$

dollar. These prices are generated randomly for all tasks present in the system. In case of symmetric schemes, we set price per resource as 75 cents[1]. The tasks specify a Quality of Service (QoS) parameter to tell how much maximum they can afford to pay for each task. Note that this QoS parameter can be directly captured in RBS as R^i_{max} while this can not be done so by NBS.

We calculate the total revenue to the CSP when the number of tasks present in the system is varied from 200 to 450. As NBS can not handle the maximum requirement, we calculate the price based on the resources allocated to each task[2]. We propose to solve this issue by choosing the number of resources $\hat{R}^*_i$ given by $\hat{R}^*_i = min\{R^*_i, R^{max}_i\}$, where R^*_i is the result of NBS optimization problem described earlier and $\hat{R}^*_i$ is the new solution for NBS.

The results for both symmetric and asymmetric cases (using the notations *NBS modified Sym* and *NBS modified Asym* respectively) are plotted in Figure 5.9. The graph also shows the pricing scheme for original NBS and RBS solutions. We can observe that, when the number of tasks present in the system are very less, original NBS allocated resources more aggressively and hence the revenue generated is more for both symmetric and asymmetric NBS schemes. If we take the modified NBS for symmetric and asymmetric bargaining, the results are similar to the RBS counterparts.

We can also observe that asymmetric scheme performs better than symmetric scheme in both cases and asymmetric scheme is more efficient when the number

[1]For an extra large standard or high-CPU on-demand instance of linux/unix, Amazon EC2 charge 0.76 cents per hour [12]

[2]In this case we can view as the users do not have any restriction to pay for all the resources allocated as it reduces task execution time

of tasks present in the system is more. This is because, when more tasks are present, the allocator has an option to allocate more resources for tasks who can afford to pay more. For example, when the number of tasks present in the system is 450, both asymmetric schemes generates a better revenue compared to their symmetric counterparts. As current CSPs follow symmetric pricing schemes, the results on asymmetric pricing approach would give adequate flexibility in managing the resources as well as generating more revenue.

5.3.3 Combined effect of deadline and pricing on resource allocation

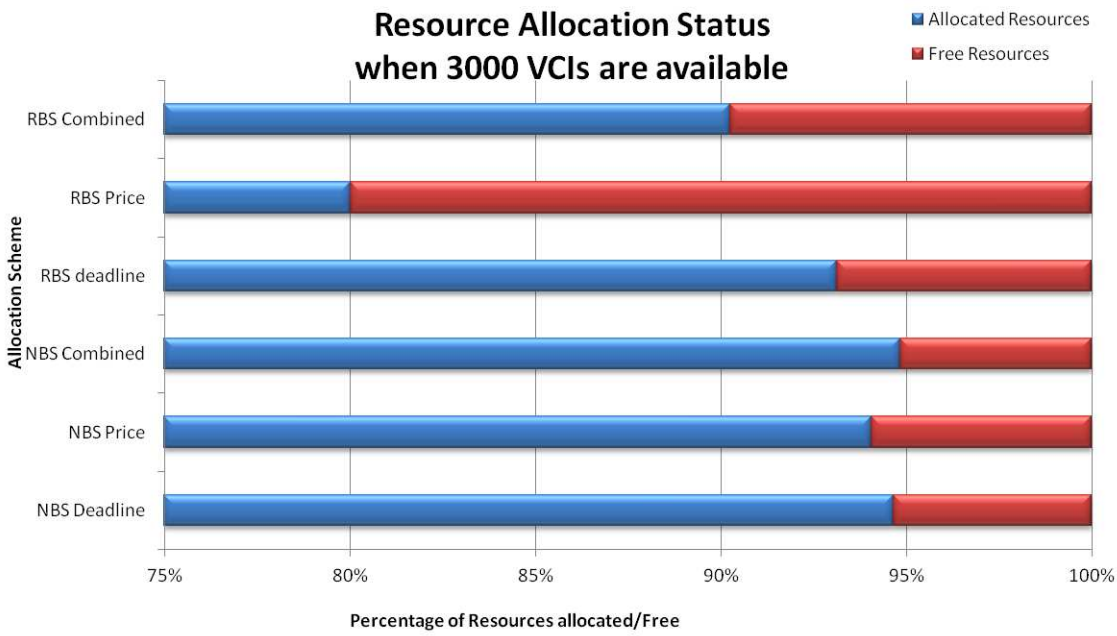


Figure 5.10: Analysis of the combined effect of pricing and deadline on resource allocation

In this subsection, we study the combined effect of deadline and pricing on

resource allocation using NBS and RBS by capturing bargaining power as a function of deadline and pricing. We generate random values for deadline and user budget for all the tasks present in the system and calculate the bargaining power using deadline, user budget and combined deadline and budget requirements and the results are plotted in 5.10.

We can observe that, in all cases, NBS aggressively allocates most of the resources and maximizes the resource utilization whereas RBS also considers the maximum resource requirements in the allocation. Calculating bargaining power using both deadline and pricing results in a resource allocation which is favorable to short deadline tasks that have higher budget the most, at the same time satisfying the requirements for all tasks. The resource utilization in RBS is highest (92%) when bargaining power is purely based on deadline and lowest when bargaining power is calculated based on price (80%). We can observe that when bargaining power is calculated based on both price and deadline, the resource utilization is around 90%. To summarize, a CSP can consider any parameter or a combination of parameters in deriving the bargaining power.

5.4 Chapter Summary

For scheduling large-scale compute-intensive applications such as workflow scheduling, often times the strategies need to be adaptable to the resource demands. In this Chapter, we precisely attempt to capture this requirement and proposed two viable solutions based on axiomatic bargaining theory (NBS and RBS) that are practically realizable. NBS maximizes the utilization of resources and guarantees proportional fairness whereas, RBS considers maximum requirement of resources

for allocation, which is useful when we want to consider the Cloud wherein tasks that are either as independent tasks or from workflow schemes arrive in a dynamic fashion. In our simulation study, we had shown how RBS effectively handles fluctuations in the resource requirements (auto-elasticity property) and real-time task arrivals up to some extent. One important observation to make is on the choice between NBS and RBS which our simulation study reveals. NBS is shown to be suitable for shorter deadline tasks whereas RBS is shown to be applicable for handling tasks of longer deadline. We also demonstrated that our schemes are adaptable to the CSPs' requirements (choice of bargaining power α and minimum and maximum resource requirements for each task) and that these schemes are shown to offer a *win-win* situation to CSPs and Cloud customers.

An important contribution of this work is in introducing asymmetric pricing scheme wherein a user is given a complete flexibility to specify his budget constraints and CSPs can attempt to maximize the revenue without sacrificing the performance. Finally we show the combined effect of task deadline and budget requirements in allocation of resources for both NBS and RBS. NBS efficiently utilizes maximum number of resources in the Cloud whereas RBS indirectly maps to an energy efficient solution by meeting the deadline with less number of resources.

Chapter 6

Design and Analysis of Broker-Mediated Cloud Aggregation and Task Scheduling Mechanisms Using Markovian Queues for Bag-of-Tasks

6.1 Introduction

While we discussed about task aggregation among Clouds offering similar types of resources in the last Chapter, we address aggregation of BoT applications among multiple Clouds having heterogeneous compute capabilities in this Chapter. Scheduling of data-intensive BoT applications from several fields including bioinformatics [76], [77], quantum optics [78] and factoring large numbers for ad-

vanced cryptography [79] on various distributed systems received huge attention in the recent past. BoT applications can be structured as a set of independent computational tasks and a task may have some instructions or data of arbitrary size and complexity. A typical example is the matching of DNA to independent known sequences [80]. Based on the results obtained from the execution of some of the tasks, other tasks may be canceled or modified.

In this Chapter, we address the aggregation of data-intensive BoT applications across multiple CSPs offering resources with different capabilities and price offers to meet some objectives such as task-deadline and/or budget requirements. Based on the number of tasks present or the amount of data to be processed, resource requirements may change with time. Further, we address the task scheduling and its effect on a particular Cloud after the task aggregation is completed. We employ the principles of *Markovian queues* for the problem under consideration.

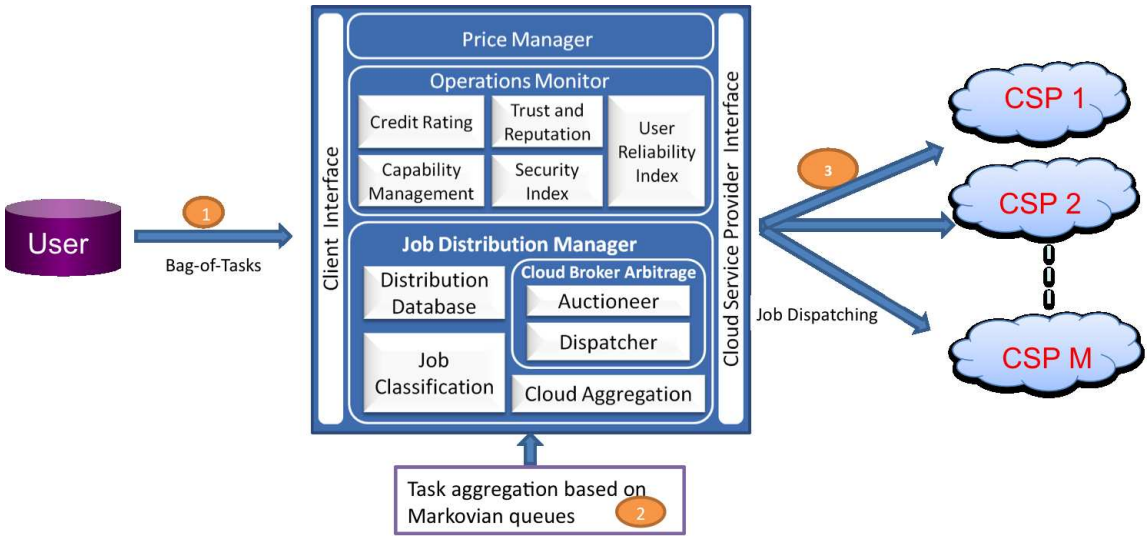


Figure 6.1: Proposed architecture for the Broker-mediated Cloud-aggregation mechanism

6.2 Proposed Multiple-Cloud Aggregation and Task Scheduling Mechanism

First we describe the markovian queue based optimization problem and derive optimal load fractions for task distribution in order to minimize task execution time. Later we also describe a heuristic algorithm to take into account the user's budget constraints. At any point of time, a user C_i can submit a BoT application request having X tasks to the Broker. Using the databases maintained by the Broker, it makes a task schedule and distributes the tasks to various CSPs. The system model is similar to the architecture discussed in Chapter 5 and is illustrated in Fig. 6.1.

6.2.1 Task distribution to minimize application completion time

Now, we derive optimal fraction of tasks to be sent out to various CSPs in order to minimize the time spent by various tasks of a BoT application in the system. If p_j is the fraction of job to be send to CSP j, then $\lambda_j = p_j\mu_0$. Further, if N_j is the number of tasks in CSP j, then,

$$N_j = p_j X \tag{6.1}$$

Further, considering each queue as an independent M/M/1 queue, it follows [81] that,

$$t_j = \frac{1}{\mu_j - p_j\mu_0} \tag{6.2}$$

Further, we can calculate total time all N_j tasks spent in the system as,

$$T_j = N_j t_j = \frac{p_j X}{\mu_j - p_j \mu_0} \tag{6.3}$$

So we formulate the optimization problem as

Minimize

$$\sum_{j=1}^{M} \frac{p_j X}{\mu_j - p_j \mu_0} \tag{6.4}$$

subject to

$$\sum_{j=1}^{M} p_j = 1 \tag{6.5}$$

$$p_j \geq 0 \forall j = 1, 2, \ldots, M \tag{6.6}$$

Temporarily relaxing the constraint (6.6) and applying Lagrange Multiplier K for the optimization problem, the augmented cost function could be written as

$$L = \sum_{j=1}^{M} \frac{p_j X}{\mu_j - p_j \mu_0} - K(\sum_{j=1}^{M} p_j - 1) \tag{6.7}$$

Now, taking the partial derivative of L with respect to each p_j and setting the derivative to zero, we can obtain,

$$\frac{\partial L}{\partial p_i} = 0 \Rightarrow K = \frac{X \mu_j}{(\mu_j - p_j \mu_0)^2} \tag{6.8}$$

Further simplifying, we obtain,

$$p_j = \frac{\mu_j}{\mu_0} - \frac{1}{\mu_0} \sqrt{\frac{X \mu_j}{K}} \tag{6.9}$$

Since, $\sum p_j = 1$, it follows that

$$\sum(\frac{\mu_j}{\mu_0} - \frac{1}{\mu_0}\sqrt{\frac{X\mu_j}{K}}) = 1 \tag{6.10}$$

and we obtain the value of K as,

$$K = \frac{X(\sum\sqrt{\mu_j})^2}{(\sum\mu_j - \mu_0)^2} \tag{6.11}$$

Substituting the value of K from Equation (6.11) in Equation (6.9), we obtain a closed form solution for the individual load fractions p_j for each CSP P_j as,

$$p_j = \frac{\mu_j}{\mu_0} - \frac{\sqrt{\mu_j}(\sum\mu_j - \mu_0)}{\mu_0\sum\sqrt{\mu_j}} \tag{6.12}$$

Thus we can calculate the number of tasks to be send out to each CSP N_j using Eq. (6.1). Note that, the values of N_j obtained need not be integer quantities. In reality, the assigned load fraction will be in terms of integer quantums. Thus, without loss of generality, we can assume that the minimum possible granularity for any N_j that can be assigned to a CSP as 1. We use integer approximation technique [82] which runs in *O(M)* to ensure that each processor gets an integer load quantum .

We illustrate this through an example. Assume $X = 500$, $M = 5$ and $\mu_0 = 70$, then various values obtained based on our results are summarized in Table 6.1. We can observe that, the constraints specified by Equations (6.5) and (6.6). Further, values of p_j's are in proportion to the service rates of various CSPs.

Table 6.1: Example to illustrate the mathematical model

CSP (j)	Service Rate (μ_j)	Job fraction (p_j)	No. of Tasks (N_j)	Unit cost (D_j)	Total task execution time (T_j)	Total Cost ($N_j D_i$)
1	75	0.21	107	0.5	1.79	53.50
2	65	0.13	66	0.4	1.17	26.40
3	77	0.23	116	0.7	1.91	81.20
4	70	0.17	86	0.6	1.49	51.60
5	79	0.25	125	0.5	2.03	62.50
Total task execution time ($max(T_j)$) and total expenditure ($\sum N_j D_i$)					2.03	275.20

Eliminating CSPs with lower resource capabilities

Note that, when we derived the optimal values for p_j's, we relaxed the constraint specified by Eq. (6.6). This means that, values of all p_j's need not be always greater than zero. For example, consider the above example with $\mu_2 = 35$ instead of 65. Then the values of p_j's obtained are summarized in Table 6.2. From Table

Table 6.2: Example to illustrate *Drop-out condition*

CSP (j)	Service Rate (μ_j)	Job fraction (p_j)
1	75	0.26
2	35	-0.05
3	77	0.28
4	70	0.22
5	79	0.30

6.2, we can see that for a CSP which has resources with lower specifications such as low service rate, the value of p_j becomes negative. In such cases, we propose to avoid such slow CSPs and recalculate the values of p_j. Avoiding CSP 2 in the above example, we obtain various values as given in Table 6.3. In this case, we obtain the values such that $\sum_{j=1}^{M} p_j = 1$ and that $p_j \geq 0 \forall j = 1, 2, \ldots, M$. We can also observe that, as tasks are distributed among fewer CSPs, the total task execution time increases.

Table 6.3: Example *Drop-out condition*: Avoiding slow CSPs

CSP (j)	Service Rate (μ_j)	Job fraction (p_j)	No. of Tasks (N_j)	Unit cost (D_j)	Total task execution time (T_j)	Total Cost ($N_j D_i$)
1	75	0.25	124	0.5	2.15	62.00
2	35	0	0	0.4	0	0
3	77	0.27	133	0.7	2.27	93.10
4	70	0.20	102	0.6	1.83	61.20
5	79	0.28	142	0.5	2.39	71.00
Total task execution time ($max(T_j)$) and total expenditure ($\sum N_j D_i$)					2.03	287.30

Table 6.4: Heuristic algorithm for task distribution based on budget requirements

```
1)  Input user budget B
2)  Calculate N_j's based on the derivations in Section 6.2.1
3)  Calculate total expenditure D_tot
4)  WHILE D_tot > B THEN
5)      Eliminate the CSP which correspond to max(D_j)
6)      Calculate all N_j's excluding this CSP
7)      Apply integer approximations to N_j
8)      Recalculate total expenditure D_tot
9)  END
10) Output the task distribution obtained at present
```

6.2.2 Task distribution based on budget requirements

Now we describe a simple, but effective heuristic to find a task distribution which can also handle the user's budget requirements in addition to the application execution time. When the optimal allocation obtained in Section 6.2.1 exceeds the user budget, we opt out the CSP which has the highest cost per resource and recalculate the optimal allocation for the minimization problem described in Section 6.2.1. We repeat this process until the total expenditure is less than or equal to the budget requirements specified by the user. The pseudo-code for this algorithm is summarized in Table 6.4.

For example, consider the example problem described in Table 6.1. The total cost for that allocation is \$275.20. Now, suppose the user budget requirements

specifies that the total expenditure shall not exceed \$240.00. So following the algorithm, we avoid CSP P_3 in first iteration and CSP (P_4) in the text iteration, we obtain the values as shown in Table 6.5.

Note that, now the total expenditure satisfies the budget requirements and hence

Table 6.5: Recomputed optimal values without P_3 and P_4 in Example1

CSP (j)	Service Rate (μ_j)	Job fraction (p_j)	No. of Tasks (N_j)	Unit cost (D_j)	Total task execution time (T_j)	Total Cost ($N_j D_i$)
1	75	0.35	176	0.5	3.49	88.00
2	65	0.26	129	0.4	2.76	51.60
3	77	0	0	0.7	0	0
4	70	0	0	0.6	0	0
5	79	0.39	195	0.5	3.77	97.50
Total task execution time ($max(T_j)$) and total expenditure ($\sum N_j D_i$)					3.77	237.10

the algorithm stops here. Further we can observe that, as we reduce the number of CSPs required for the task distribution, the total application execution time increases.

6.3 Task scheduling within a Cloud environment

So far, we have described a model to aggregate a set of BoTs among multiple Clouds with different types of resources. Now, we take a closer look at the scheduling of tasks among datacenters within a Cloud after the aggregation is performed. The novelty of the scheduler lies in segregating tasks into two different priority queues based on whether a task is *uncertain*[1] or *certain*, assigns higher priority to *certain* tasks and employs a novel scheduling mechanism to minimize the task computation time, resource usage as well as the monetary cost for the task execution.

[1]An *uncertain* task may be canceled or modified at a later point of time whereas a *certain* task will not be modified once submitted for execution.

The Cloud $\mathbb{C}$ under consideration consists of a number of datacenters (DCs) with $\mathbb{C} = \{\mathbb{DC}_1, \mathbb{DC}_2, \cdots, \mathbb{DC}_i, \cdots\}$, where datacenter $\mathbb{DC}_i = \mathbb{H}_i \cup \mathbb{S}_i \cup \mathbb{L}^i_{LAN} \cup \mathbb{L}^i_{WAN}$. $\mathbb{DC}_i$ is the ith datacenter participating in $\mathbb{C}$. Each datacenter, contains a set $\mathbb{H}_i$ of *physical hosts*, a set of $\mathbb{S}_i$ of storages for storing data, a set $\mathbb{L}^i_{LAN}$ of local area network (LAN) links used for communicating among the hosts in $\mathbb{H}_i$, and a set $\mathbb{L}^i_{WAN}$ of wide area network (WAN) link with other datacenters. For modeling purposes, specific network topologies or technologies are not considered, and we ignore the impact of the internal network topology on the speed of both inter-physical host or inter-VM data transfers. This is because, we focus on task scheduling and designing a datacenter network to improve the data transmission efficiency for data-intensive communication in datacenters itself is a hot research area ([83] and [84] to quote few). Communication links between physical hosts are assumed to be contention free to accommodate the deterministic nature of scheduling, and communication between them is via I/O channels thereby allowing concurrent computation and communication.

It is interesting to note that the above model can handle not only the subset of tasks dispatched by the Broker, but also tasks submitted directly by independent BoT applications. Thus the dispatcher itself can be a Broker within one Cloud to perform the task scheduling among multiple datacenters. Hence, without any loss of generality, we consider task scheduling of independent BoT applications[1] in the remaining Chapter.

A *virtual cluster* (VC) consists of multiple VMs (combined together based on user requirements), that are logically connected together over one or more phys-

[1]Tasks submitted by the Broker can be considered as a set of tasks within a BoT application

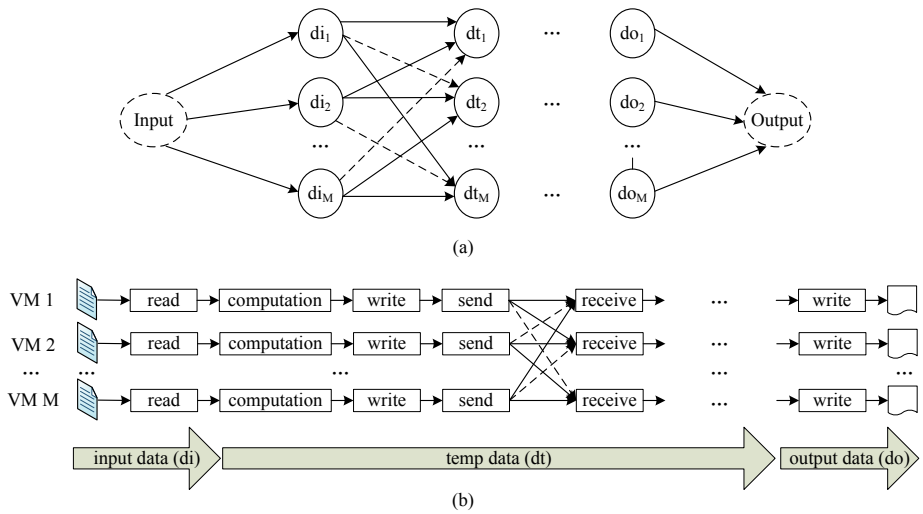


Figure 6.2: The system model. (a) Data from a BoT application arriving at the Cloud system can be assigned to M VMs with input data di, where, di_1, di_2, and di_M are belong to task n_1, n_2, and n_M, respectively. This is, task n_1, n_2, and n_M are assigned to VM VM_1, VM_2, and VM_M, respectively. In the same way, dt is the temp data created by Cloud system, and do is the output data of Cloud system. (b) A map-reduce example.

ical servers. The communication overhead between two tasks scheduled on the same VM is assumed to be zero. Each VC is an administrative domain that has its own master and slaves.

Our system model (Fig. 6.2) assumes that input data is distributed across all participating VMs in a VC, and that each VM retrieves its initial input from local storage. The model also accounts for output data replication, assuming the common strategy of storing the first replica on the local disks and sending the others over the network to other physical hosts.

We denote the amount of input data for VM_j as di_j and output data for VM_j as do_j. The amount of temporary data produced or consumed by computation for VM_j is denoted as dt_j. For many map-reduce applications, the mapper will

implement some form of selection or filtering, and the reducer will perform aggregation. A communication cost is only required when two tasks are assigned to different VMs. In other words, the communication cost when tasks are assigned to the same VM can be ignored.

Different BoT applications may have different requirements in categorizing their tasks into *certain* or *uncertain* tasks. Hence, without any loss of generality, we assume that the users or the Broker indicate whether each task is *certain* or *uncertain* upon submission based on their prior knowledge about the tasks. We also assume that, *uncertain* tasks may be canceled or modified, but *certain* tasks will never be canceled. This is done to make sure that, our scheduler can seamlessly work for a wide variety of applications.

The scheduler decides the order in which the tasks need to be executed. The scheduler communicates decisions to the operating systems running on the VC resources which handles the details of running tasks on the VMs and provides task statistics to the scheduler. If a *certain* task is executed, then the task output is supplied to the requesting user. If an *uncertain* task is executed, then the task output is stored in a location isolated from the rest of the system until it becomes *certain* or *canceled*. A task could be marked *uncertain* by its submitter and assigned a real number between 0 and 1 indicating the probability of eventual need at the time of submission. We also assume that the system does not allow to promote an *uncertain* task to a *certain* task, or to cancel a *certain* task. We define three parameters for evaluating the performance of our proposed scheduling strategy— Makespan, Monetary cost and Resource Usage Index.

6.3.1 Makespan

The makespan of an application is the time elapsed once the application is submitted until its execution is completed [85]. It is a natural metric for the application performance due to its ability to reflect a users view of how long an application takes to complete its execution.

As users are generally more interested in the *certain* task's execution time, we define a new metric, M_c as the makespan of *certain* tasks. It denotes the time duration since a task has been submitted till the necessary outputs are obtained. M_c accrues only after a user asks for output from some *certain* tasks that may have been submitted much earlier and thus measures the time that a user actually waits for output. M_c differs from the actual makespan in the sense that M_c is calculated only based on *certain* tasks whereas actual makespan computation includes all tasks including *uncertain* tasks [86]. We define the M_c for $\mathbb{N}$ BoT applications as $M_c = max\{T_1^0, T_2^0, \ldots, T_K^0\}$ where, K is the number of VCs. Note that, the execution time of *uncertain* tasks are not considered in this case.

6.3.2 Monetary Cost

Our computation cost model assumes that computation time unit, expressed in hours, are identical for all VMs. Each VM, however, has its own cost per hour, expressed in dollars. CSPs compute the time for which the VCs are used by the tasks. The billing starts when the VC is ready and is allocated to the requested application's first task and ends when the last task completes its execution. In practice, if a VC is not released to CSPs then the user will be charged even if the VC is idle. Thus, the total computational monetary cost of all BoT applications

$Cost_{total}$ for all K VCs can be calculated as:

$$Cost_{total} = \sum_{k=1}^{K} max\{T_k^0, T_k^1\} \cdot c_k \cdot n_k \tag{6.13}$$

where T_k^0 and T_k^1 are the task execution time for *certain* and *uncertain* tasks respectively and c_k is the price per VM. In other words, c_k is the monetary cost/computation time unit of VM_k expressed in dollars. For example, let let $max\{T_k^0, T_k^1\}$ be 30000 seconds and there are 64 VMs. If the CSP charges $\$0.1/hour$ for one VM in the VC_k, the actual monetary cost is $\lceil 30000/3600 \rceil \times 64 \times 0.1 = \$57.6$.

6.3.3 Resource Usage Index (RUI)

The goal of the proposed elastic scheduling scheme is to improve the usage of the resources in the Cloud and performance of submitted bag-of-tasks. In order to calculate the resource usage, we define another metric RUI, ψ as the ratio of the used resource capacity of the VC to the total resource capacity of that VC and is used to show the utilization of Cloud resources. For the same value of m_k, higher value for RUI means higher resource utilization for certain tasks, for the same percentage of uncertain tasks and higher RUI means more uncertain tasks are scheduled.

6.3.4 The Queuing Model for Task Scheduling

We assume there are N BoTs (t_1, t_2, ..., t_N) and each BoT t_j has it own data di_j. Further there are K VCs containing M VMs that store a total of D different data di_1, di_2, ..., di_D, where $D \gg M$. We use $\mathbb{DI}$ to denote the set of input data

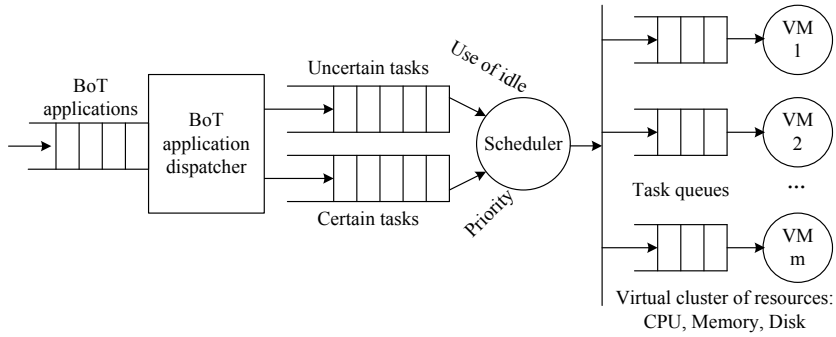


Figure 6.3: The scheduling model dispatches BoTs to a virtual cluster for parallel execution in a Cloud platform. This model can handle not only the subset of tasks dispatched by the Broker, but also tasks submitted directly by independent BoT applications. Thus the dispatcher itself can be a Broker within one Cloud (private or public Cloud) to perform the task scheduling among multiple datacenters.

in the system. Since each data di_j may have one or more replicas in practice, each object may have one or more replicas that are stored in different VMs and we use $\mathbb{V}di_j$ to denote the set of VMs that store the same data di_j. Clearly, if there is no replica for di_j, $|\mathbb{V}di_j| = 1$ holds for each di_j in $\mathbb{V}di_j$. We define VM_j^p as the primary VM for di_j, which stores one replica of data di_j and takes charge of the scheduling on the data di_j.

BoT applications are assigned to the Cloud system $\mathbb{C}$ and let λ be the total task arrival rate in the entire Cloud system. If VC_k is scheduled with probability p_k, then, $\sum_{k=1}^{K} p_k = 1$. Our method attempt to minimize user expenditure, resource usage and task execution time by appropriate scheduling of the BoTs. *Uncertain* and *certain* tasks are segregated into two queues by the BoT application dispatcher and *certain* tasks are assigned higher priority, resulting in better makespan. *Uncertain* tasks are scheduled whenever VMs are idle. The queuing model of the BoT scheduling is shown in Fig. 6.3. If a task is canceled, it may

affect other tasks in the queue. Further, efforts in transferring the input data to respective VM as well as the computation time involved till the task gets canceled are wasted. Hence, we should postpone or avoid to schedule such tasks and give higher priority to *certain* tasks.

We assume that in a virtual cluster VC_k, the arrival rate of *certain* task and *uncertain* task are λ_k^0 and λ_k^1, respectively. As shown in Fig. 6.3, VC_k has one queue to store *certain* tasks and another one to store *uncertain* tasks. When a task arrives at the VC, it waits at the corresponding queue which follows a non-preemptive First Come First Serve (FCFS) queuing discipline. Note that, in a VC, there may be several VMs and each VM may have different computation and I/O capacity.

For ease of simplicity, we assume that the aggregate computation and I/O bandwidth capacity on VC_k are $Ccom_k$ and Cio_k, respectively. The scheduling can be done based on the available compute and I/O bandwidth capacity. Due to the similarity in disk bandwidth and network resource considerations, we only consider disk bandwidth for scheduling and we assume that there is sufficient network bandwidth available between VMs.

For a BoT application, the task arrival rate for VC_k is $\lambda_k = p_k \cdot \lambda$. When a task t_i is assigned to a VC, $Ccom_i$ and Cio_i should be assigned to this task to guarantee the task execution. We can obtain the expected computation and I/O bandwidth capacity to execute a task t_i in the system as:

$$E[Ccom_i] = \frac{\sum_{j=1}^{K} p_j \cdot Ccom_j}{K} \tag{6.14}$$

$$E[Cio_i] = \frac{\sum_{j=1}^{K} p_j \cdot Cio_j}{K} \tag{6.15}$$

When a task is arriving at VC_k, scheduler determines appropriate VM, VM_i for its execution if that task can be scheduled. The task is then forwarded to the local queue of VM_i. From Eq.(6.14) and Eq.(6.15), we can calculate the expected number of tasks which can be served in a service round on VC_k as $e_k{=}min\{\lfloor\frac{Ccom_k}{E[Ccom_i]}\rfloor,\lfloor\frac{Cio_k}{E[Cio_i]}\rfloor\}$. We use μ_k to denote the service rate of VC_k and model a VC as $M/M/m$ queueing system. Besides the *uncertain* tasks, VC_k needs to accept those *certain* tasks that have higher priorities.

In order to guarantee shorter makespan ($T_{makespan}$) for BoT applications, we adopt priority-based scheduling policy in our system. The basic idea is that *certain* tasks can use all of the VMs and *uncertain* tasks can use only up to m_k VMs, where $0 < m_k \leq n_k$, where n_k is the number of VMs present in k-th VC. We assume that when tasks are assigned to VC_k, if there is no available VM at that moment, scheduler will block *uncertain* tasks in the *uncertain* queue, but schedule *certain* tasks in the *certain* queue and/or let them wait until some VMs are released. Here, we assume that the length of both queues are infinity.

Given a state space of the system based on the above scheduling policy, we can identify those states where a *certain* task will wait in the *certain* queue or an *uncertain* task will wait (or be blocked) because of the capacity constraints or the priority. According to this information, we can compute the execution time of tasks. Let Φ_k be the state space of VC_k that is defined by $\Phi_k = \{(l_0, l_1) \mid l_0, l_1 \geq 0\}$, where l_0 and l_1 are the expected number of *certain* and *uncertain* tasks respectively. We can divide Φ_k into four sub-space as shown in Table 6.6 with $\Phi_k = \Phi_k^i \cup \Phi_k^{ii} \cup \Phi_k^{iii} \cup \Phi_k^{iv}$. For example, for a VC_k with $n_k = 128$ and $m_k = 32$, VC_k

can allocate only up to 32 VMs for *uncertain* tasks and up to 128 VMs for *certain* tasks. We can observe that waiting states for *certain* tasks and *uncertain* tasks will be, $\Phi_k = \{(l_0, l_1) \,|(128 - l_1) < l_0, l_1 \leq 32\}$. This is the case when $\Phi_k = \Phi_k^{iii}$ in Table 6.6. Using Little's Theorem [81], we can obtain the equilibrium probability

Table 6.6: Sub-space of Φ_k.

Symbol	**Meaning**
$\Phi_k^i = \{(l_0, l_1) \mid l_0 \leq n_k - m_k,\ l_1 \leq m_k\}$	No tasks are waiting in both queues
$\Phi_k^{ii} = \{(l_0, l_1) \mid l_0 \leq n_k - m_k,\ l_1 > m_k\}$	Some *uncertain* tasks are blocking in the *uncertain* queue and no task is waiting in the *certain* queue
$\Phi_k^{iii} = \{(l_0, l_1) \mid l_0 > n_k - m_k,\ l_1 \leq m_k\}$	No tasks are waiting in the *uncertain* queue
$\Phi_k^{iv} = \{(l_0, l_1) \mid l_0 > n_k - m_k,\ l_1 > m_k\}$	Some tasks waiting (blocking) in both queues

of l_0 *certain* tasks and l_1 *uncertain* tasks in the system in the state (l_0, l_1) of Φ_k^i, $p_k(l_0, l_1)$ as:

$$p_k(l_0, l_1) = p_k(0, 0)\frac{((n_k - m_k)\rho_0)^{l_0}}{l_0!}\frac{(m_k\rho_1)^{l_1}}{l_1!} \tag{6.16}$$

where, $\rho_0 = \frac{\lambda_k^0}{(n_k - m_k)\mu_k}$, $\rho_1 = \frac{\lambda_k^1}{m_k\mu_k}$, and $p_k(0, 0)$ is the probability that there is neither *certain* nor *uncertain* tasks under service in VC_k.

Similarly, the system in the state (l_0, l_1) of Φ_k^{ii} has the following product form:

$$p_k(l_0, l_1) = p_k(0, 0)\frac{((n_k - m_k)\rho_0)^{l_0}}{l_0!}\frac{m_k^{m_k}\rho_1^{l_1}}{m_k!} \tag{6.17}$$

The system in the state (l_0, l_1) of Φ_k^{iii} has the following product form:

$$p_k(l_0, l_1) = p_k(0,0)\frac{(n_k - m_k)^{(n_k - m_k)}\rho_0^{l_0}}{(n_k - m_k)!}\frac{(m_k\rho_1)^{l_1}}{l_1!} \tag{6.18}$$

The system in the state (l_0, l_1) of Φ_k^{iv} has the following product form:

$$p_k(l_0, l_1) = p_k(0,0)\frac{(n_k - m_k)^{(n_k - m_k)}\rho_0^{l_0}}{(n_k - m_k)!}\frac{m_k^{m_k}\rho_1^{l_1}}{m_k!} \tag{6.19}$$

We can calculate $p_k(0,0)$ using Eq.(6.16) to Eq.(6.19) and the condition $\sum_{l_0+l_1=0}^{\infty} p_k(l_0, l_1) = 1$ as,

$$\begin{aligned} p_k(0,0) = & [\textstyle\sum_{l_0=0}^{n_k-m_k-1} \frac{((n_k-m_k)\rho_0)^{l_0}}{l_0!} + \frac{((n_k-m_k)\rho_0)^{n_k-m_k}}{(n_k-m_k)!(1-\rho_0)}]^{-1} \cdot \\ & [\textstyle\sum_{l_1=0}^{m_k-1} \frac{(m_k\rho_1)^{l_1}}{l_1!} + \frac{(m_k\rho_1)^{m_k}}{m_k!(1-\rho_1)}]^{-1} \end{aligned} \tag{6.20}$$

In our proposed policy, when the system is busy, a newly-arriving *uncertain* task may be blocked in the *uncertain* queue, however, a *certain* task will wait in the *certain* queue. Since *certain* tasks have higher priority than *uncertain* tasks, *certain* tasks will be serviced by all the VMs whenever VMs are available in VC_k. In order to analyze the state in Φ_k^{ii}, Φ_k^{iii} and Φ_k^{iv}, we shall focus on the boundary states of Φ_k^{i}. Here, we use $p_k(\phi)$ to denote the probability that all VMs in the VC_k are in use but there is neither *certain* nor *uncertain* tasks are waiting in both queues. Then $p_k(\phi)$ can be calculated as:

$$p_k(\phi) = \textstyle\sum_{l_1=0}^{m_k} p_k(n_k - l_1, l_1)$$

substituting Eq.(6.16), we obtain,

$$p_k(\phi) = \sum_{l_1=0}^{m_k} p_k(0,0) \frac{((n_k-l_1)\rho_0)^{n_k-l_1}}{(n_k-l_1)!} \frac{(m_k\rho_1)^{l_1}}{l_1!} \tag{6.21}$$

Let $p_k(Q)$ denote the probability that a newly arrived *certain* task will find all VMs busy and will be forced to wait in *certain queue*. Then,

$$p_k(Q) = \sum_{l=n_k-m_k}^{\infty} \frac{p(\phi)(n_k-m_k)^{n_k-m_k}\rho_0^l}{(n_k-m_k)!} = \frac{p(\phi)((n_k-m_k)\rho_0)^{n_k-m_k}}{(n_k-m_k)!(1-\rho_0)} \tag{6.22}$$

From Eq.(6.22), we can obtain that the expected number of tasks waiting in the *certain* queue is:

$$Q_k^0 = p_k(Q)\frac{\rho_0}{1-\rho_0}$$

In the same way, we can find out the probability $p_k(B)$ that a newly arrived *uncertain* task will find all VMs, reserved for *uncertain* task, busy and will be blocked/forced to wait in the *uncertain* queue:

$$p_k(B) = \sum_{l=m_k}^{\infty} \frac{p(\phi)m_k^{m_k}\rho_1^l}{m_k!} = \frac{p(\phi)(m_k\rho_1)^{m_k}}{m_k!(1-\rho_1)} \tag{6.23}$$

and the expected number of *uncertain* tasks waiting in the *uncertain* queue is:

$$Q_k^1 = p_k(B)\frac{\rho_1}{1-\rho_1}$$

Now, we discuss the task execution time in the system. If μ_k is the service rate for cluster VC_k, using Little's Theorem [81], Eq.(6.23) and Eq.(6.24), we obtain

the execution time T_k^0 and T_k^1 for *certain* and *uncertain* tasks respectively as,

$$T_k^0 = \frac{Q_k^0}{(n_k - m_k)\mu_k} \tag{6.24}$$

$$T_k^1 = \frac{Q_k^1}{m_k \mu_k} \tag{6.25}$$

This implies that, task execution time of *certain*(*uncertain*) tasks depends on the number of VMs available for execution, the expected number of tasks waiting in the *certain*(*uncertain*) queues and the service rate of the Virtual Cluster VC_k. Task execution time of *certain* tasks also depends on the number of VMs that can be allocated to *uncertain* tasks at any point of time.

6.4 Performance Evaluation and Discussions

6.4.1 Performance analysis of multiple-Cloud aggregation mechanism

In this section, we perform rigorous simulation experiments to evaluate the performance of the proposed task aggregation scheme. We conduct experiments to evaluate the task execution time, total expenditure and the task distribution pattern for large scale Broker-mediated multiple Cloud environments. The simulation parameters are summarized in Table 6.7. We analyze the task execution time for various user budget requirements and the results are plotted in Figure 6.4. We can observe that, when no budget requirements are specified, all the 10 CSPs are used and the total tasks execution time is 2.79 units. As the user

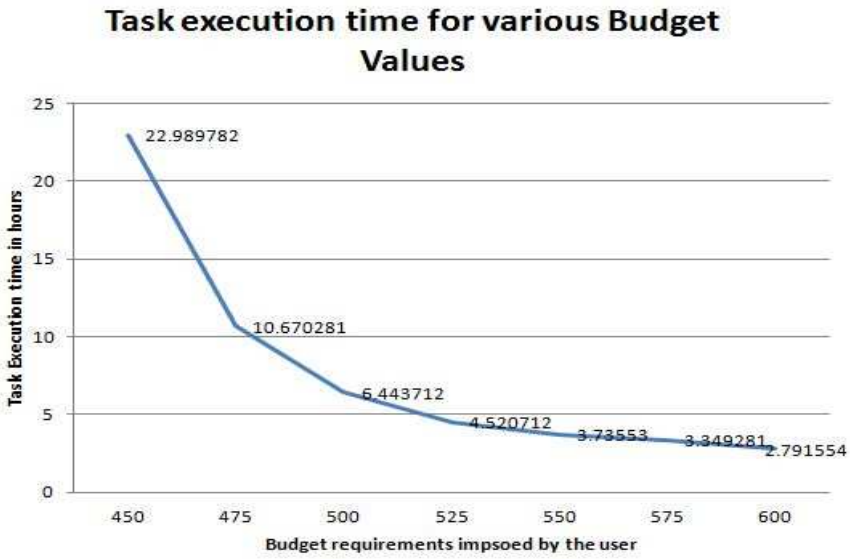


Figure 6.4: Task Execution Time

imposes higher budget restrictions, the total task execution time increases. This is because, when higher budget restrictions are imposed, tasks are distributed to fewer CSPs and hence takes more time to complete the same tasks. Now, we compare the total user expenditure for various budget requirements imposed by the user. From Fig. 6.5, we can observe that, in all cases, the expenditure is lower than the user budget requirements. We can also observe that, a minimum cost is involved when all the tasks are submitted to only one CSP which offers

Table 6.7: Major simulation parameters

Parameter	Value	Parameter	Value
M	10	X	1000
B	[450,600]	μ_0	100
μ_1	61	D_1	0.5
μ_2	67	D_2	0.4
μ_3	77	D_3	0.7
μ_4	60	D_4	0.6
μ_5	79	D_5	0.5
μ_6	54	D_6	0.75
μ_7	74	D_7	0.55
μ_8	58	D_8	0.6
μ_9	72	D_9	0.65
μ_{10}	64	D_{10}	0.7

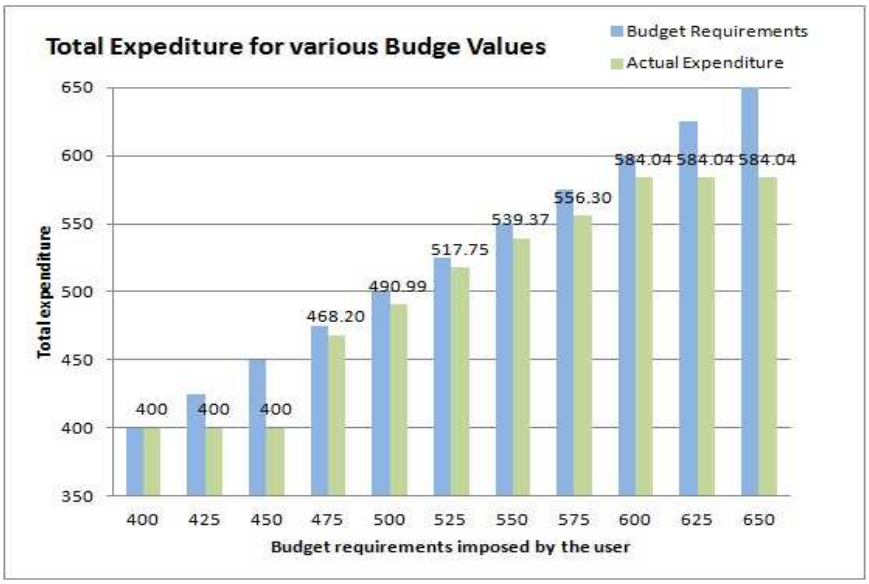


Figure 6.5: Expenditure

the least price for the resource and the maximum expenditure can be determined based on the task distribution derived from our optimization problem. In this

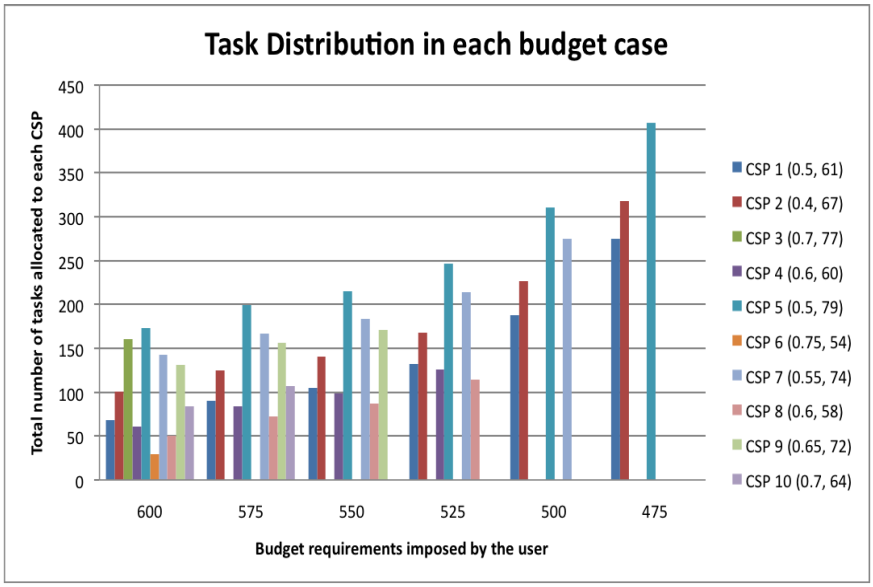


Figure 6.6: Task distribution

section, we analyze the distribution of tasks to different CSPs based on various

user budget requirements and results are plotted in Figure 6.6. In this figure, for example *CSP1(0.5,61)* means that $CSP1$ has a price offer 0.5 and a service rate 61. From Figure 6.6, we can understand that, when a user is flexible with his/her budget, tasks are distributed evenly among the CSPs available for the task execution. But as the user imposes more budget restrictions, the tasks are distributed among fewer CSPs and hence more resources are used from these CSPs to complete the same number of tasks.

6.4.2 Performance analysis of the task scheduling strategy within a Cloud environment

Now we evaluate the performance of the task scheduling within a Cloud. Our simulator models a number of BoT applications which are submitted to a number of VCs in the Cloud. The number of applications is varied across simulations. Refer Table 6.8 for various simulation parameters used in our experiments in this section. A uniform distribution (uni.) is described by 'lower bound to upper bound', where the bound is specified by a range varied across runs. An exponential distribution (exp.) is described by its mean, where the mean is specified by a range varied across runs. A step determines the distance between each sample in the range of its parameter. Important parameters considered in the experiments are:

1. *Uncertain task proportion:* It indicates the ratio of *uncertain* tasks with respect to the total number of tasks in a BoT application. Each application has an *uncertain* task proportion randomly chosen following a continuous uniform distribution. For example, if the *uncertain* task proportion is set

to 0.2, then 20% of the tasks are *uncertain* in the applications.

2. *Write back time:* It indicates the time required by the tasks to write the output back to the external device such as disk. The write-back time values are also generated randomly following exponential distribution. Note that, the task runtime is the sum of computation time and write back time.

Table 6.8: Parameters used in the experiments in Section 6.4.2

Parameter	**Setting**	**Step**	**Sample**
number of app.	1	-	-
number of BoTs per app.	1000	-	-
number of tasks per BoT	1 to 5 (uni.)	1	5
uncertain task proportion	30%	-	1
computation time (s)	15 to 180 (exp.)	15	13
write back time (s)	15 to 90 (exp.)	15	7
number of VMs	64	-	1
number of app.	5 to 20	5	4
number of BoTs per app.	20 to 80 (uni.)	20	4
number of tasks per BoT	1 to 5 (uni.)	1	5
uncertain task proportion	10% to 30%	10%	3
computation time (s)	15 to 180 (exp.)	15	13
write back time (s)	15 to 90 (exp.)	15	7
number of VMs	64	-	1
number of app.	100,1000,2000, 3000,4000,5000	-	-
number of BoTs per app.	2000 to 10000 (uni.)	1000	9
number of tasks per BoT	1 to 5 (uni.)	1	5
uncertain task proportion	10% to 30%	10%	3
computation time (s)	30 to 360 (exp.)	30	13
write back time (s)	30 to 180 (exp.)	30	7
number of VMs	64, 1024	-	2

Comparison between simulation and theoretical analysis

We simulate our model in a realistic way in order to reflect the uncertain behavior of tasks in BoT applications. We made several runs of our simulation with the same parameters but different random number seeds for generating random variables to confirm that the variability was sufficiently small to make the results dependable. Here, we compare the values of M_c and RUI obtained based on our theoretical analysis as well as based on our simulation results. For that purpose,

Table 6.9: Comparison between simulation and theoretical analysis.

Items	Theoretical analysis	Simulation
M_c	10237.079	9970.914
RUI	0.923813	0.922058

we consider a BoT application with 1000 BoTs and we set the uncertain task proportion to be 30%. Further, we set the task computation time and write back time to be exponentially distributed with mean $360s$ with he number of VMs set to 64 and the maximum number of VMs allocated to uncertain tasks m_k is set to 13. The values obtained for our theoretical analysis described in Section 6.3.4 and simulated metrics are summarized in Table 6.9. We can observe that, the values are very close to each other with only less than 3% difference for M_c and less than 0.2% difference for RUI.

Experimental Results for task scheduling within a Cloud

We conduct experiments to analyze various metrics such as makespan, monetary cost and resource usage for our elastic scheduling scheme and compare the values with traditional batch scheduling scheme which follows FCFS scheduling. In our experiments, we vary the number of VMs kept aside for the *uncertain* tasks. For example, 'elastic(6/58)' means, there are 64 VMs, with a maximum of 13 VMs for *uncertain* tasks. *Certain* tasks can use up to 64 VMs because all the VMs can be used by *certain* tasks if there are tasks in the *certain* queue.

Our workload model is similar to the workload described in [87]. In our system model, we assume that the VMs are grouped into clusters. The VMs may have different performance across clusters, but within the same cluster they are homogeneous. The workload of the system consists of applications submitted by user(s); each of the applications consists of a number of Bag-of-Tasks which is a

bag of unordered tasks (possibly only one). Upon their arrival into the system, the BoTs are queued in certain queue or uncertain queue respectively, waiting for available VM on which to be executed. Once started, BoTs run to completion, so we do not consider BoT preemption or BoT migration during execution. Instead, tasks can be canceled when all certain BoTs are finished. Please note that, in our application model, the ranges used in simulating BoT applications and tasks were motivated by the scenarios, such as computer animated scenes in computer-animated film [88].

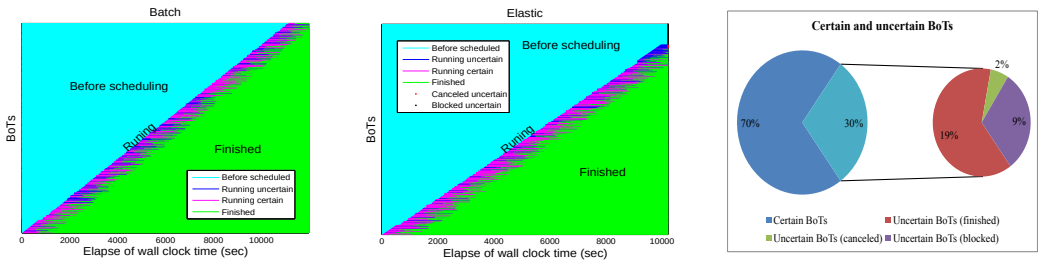
In [87], the user patterns and the user submissions are considered as Zipf distribution and, the BoT arrival patterns are modeled in two steps: first the inter-arrival time (IAT) between consecutive BoT arrivals during peak hours, and then the IAT variations caused by the daily submission cycle. We have given more importance to the scale of applications and do not consider the submission pattern by the user. So we consider the user submission to follow as a more generic uniform distribution. Currently, we consider a BoT, either all are certain tasks or all are uncertain tasks. So the certain tasks queue includes some BoTs any one of which may consist of a number of certain tasks. Also, the uncertain tasks queue includes some BoTs any one of which may consist of a number of uncertain tasks. In our model, the tasks in a BoT are unordered, which is different [87], which assumes that the tasks in a BoT are processed in a sequential manner. So, in our model, we do not consider the intra-BoT characteristics defined by reference [87].

Performance evaluation for an application with 1000 BoTs

To show the difference between *batch* and *elastic* schedule strategies, we consider batch and elastic execution process of a BoT application with 1000 BoTs as illus-

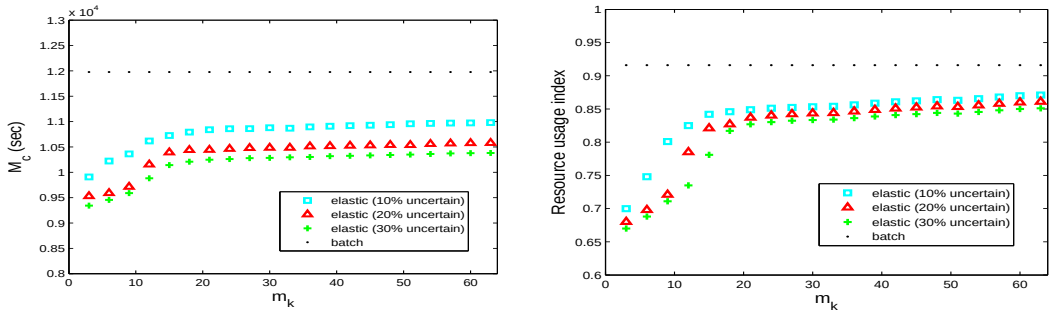
trated in Fig.6.7. In Fig.6.7(a) and Fig.6.7(b), the x-axis represents the elapsed time in seconds, and each horizontal lines in the chart represents different tasks in a BoT application. Within each line, *Before scheduled*, *Running uncertain*, *Running certain* and *Finished* represent the period before the BoT is scheduled, when the uncertain BoT is running, when the certain BoT is running and the period after the BoT is complete respectively. Similarly, *Canceled uncertain* and *Blocked uncertain* represent the point when the uncertain BoT is canceled, and when it is blocked and is not scheduled. Fig.6.7(c) shows that, for *elastic* strategy, 19% uncertain BoTs complete their execution, 2% uncertain BoTs are canceled and 9% uncertain BoTs are blocked. This means, our scheduling strategy is able to prioritize and avoid execution of those blocked and canceled tasks which is the root cause of the large makespan gap between *batch* and *elastic*.

In order to analyze the effect of the number of m_k on the performance of BoT application with certain and uncertain tasks, we plot M_c for various values of m_k for three different proportions of uncertain tasks. Fig. 6.8 shows the execution of a BoT application with 1000 BoTs on batch scheme, and elastic scheme for



(a) (b) (c)

Figure 6.7: The execution of a BoT application with 1000 BoTs (number of VMs is 64, m_k is 13, uncertain task proportion is 30%). (a) Batch strategy; (b) Elastic strategy; (c) Pie chart of certain and uncertain BoTs for elastic execution (as shown in (b)).



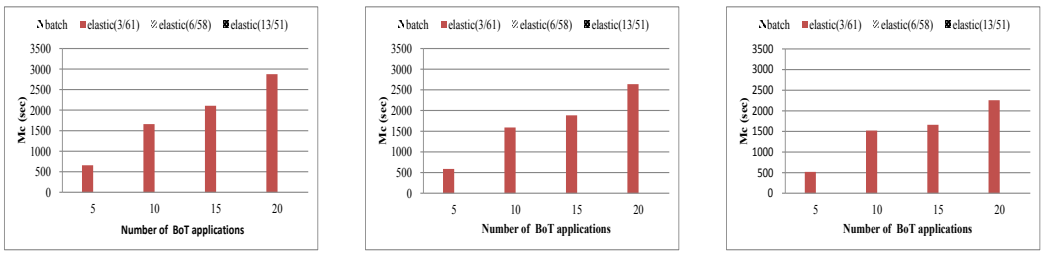
(a) (b)

Figure 6.8: The effect of the number of m_k for makespan M_c and resource usage index ψ (number of VMs is 64). (a) makespan M_c; (b) resource usage index ψ.

different M_c. We can observe that, higher proportion of uncertain tasks gives a better makespan M_c with the same m_k. Further, it is interesting to observe that the proportion of uncertain tasks associates with m_k for M_c and RUI metrics: There exists a turning point in the growth curve and turn up when the proportion of uncertain tasks and the value of m_k/n_k are approximately equal.

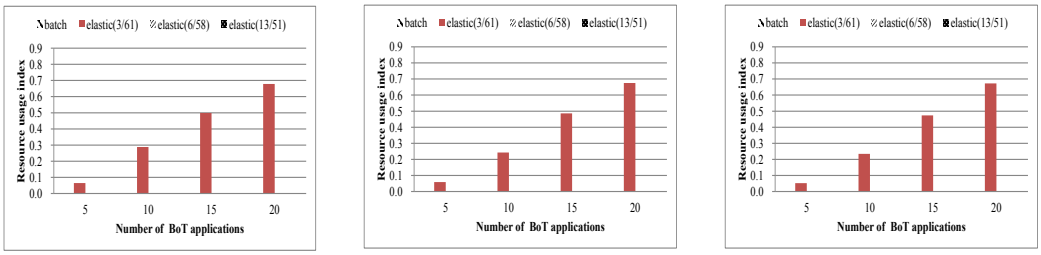
Performance evaluation with 20 BoT Applications and 64 VMs

Fig. 6.9 shows the effect of the number of applications on elastic scheme, and batch scheme for M_c. We plot the makespan of *certain* tasks M_c for various values



(a) (b) (c)

Figure 6.9: The effect of the number of applications for makespan M_c (number of VMs is 64). (a) 10% uncertain task proportion; (b) 20% uncertain task proportion; (c) 30% uncertain task proportion.



(a) (b) (c)

Figure 6.10: The effect of the number of applications for resource usage index ψ (number of VMs is 64). (a) 10% uncertain task proportion; (b) 20% uncertain task proportion; (c) 30% uncertain task proportion.

of m_k (i.e. the maximum number of VMs that can be allocated for *uncertain* tasks, $m_k = 3, 6, 13$) for three different proportions of uncertain tasks present in the applications. We can observe that our scheme always gives a better makespan M_c compared to the batch scheme. Further, it is interesting to observe that amount of uncertain tasks has no influence on M_c in case of *batch* scheme. However, as the amount of uncertain tasks increase, our scheme improves M_c.

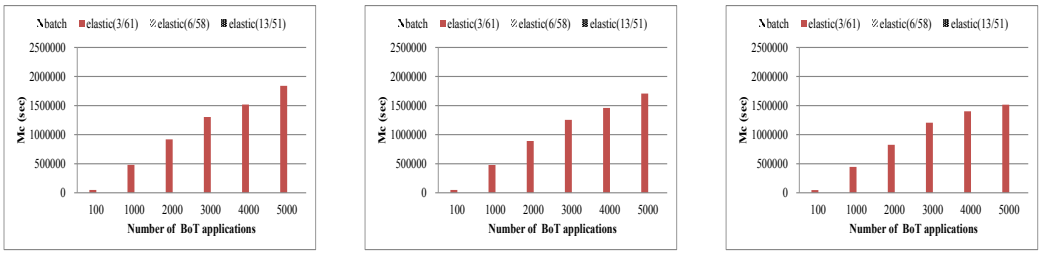
We can observe an improvement of 25.2% for M_c compared to the traditional scheme when the amount of uncertain tasks present is 30%. It means that users having higher amount of uncertain task proportion will be able to benefit more from our scheme in order to reduce their overall task execution time. In most cases, our scheme performs better compared to the batch scheduling scheme. This is because, waiting time for *certain* tasks have been reduced considerably with the prioritization of tasks. At the busiest part or low *uncertain task proportion*, both schemes begin to converge because the *certain* task queue of the elastic scheduler is never empty.

Fig. 6.10 plots the *resource usage index*, ψ for various number of BoT applications in the Cloud. When the value of ψ is large, more resources are used

for completing the execution of BoT applications. Further, as the uncertain task proportion increases, the values of ψ is better for our scheme, whereas its value remains the same for *batch* scheme in all cases. It is also interesting to note that, our scheme shows an average improvement of 28.13% for ψ compared to *batch* scheme when 30% of the tasks present are *uncertain*. We obtain the best case improvement of 52.55% and minimum improvement of 22.25% for ψ when there are 5 and 20 applications respectively. The higher value of ψ in case of traditional scheme is because, it does not distinguish between *certain* and *uncertain* tasks and uses the resources for *uncertain* tasks which are later modified or canceled.

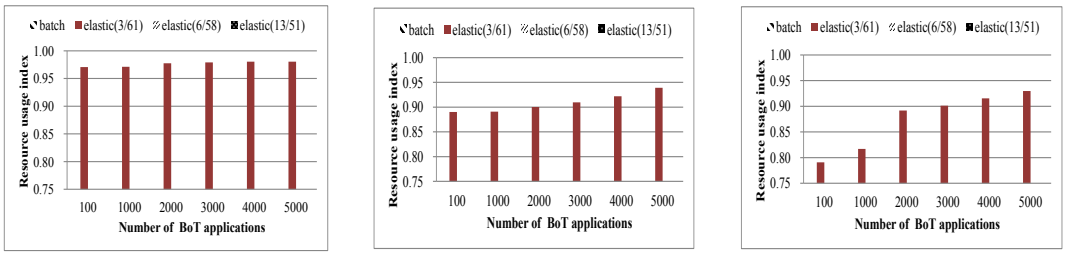
Large scale Performance Analysis with 5000 BoT Applications

Now we conduct experiments to validate our findings for large-scale BoT applications. The results obtained for M_c and ψ for up to 5000 BoT applications, are plotted in Figures 6.11 and 6.12 respectively. We can observe that, our scheme outperforms the traditional *batch* scheme even when we scale up the number of BoT applications in the system to 5000. Further, it is evident from the figures that, when lesser number of VMs are reserved for *uncertain* tasks, our scheme



(a) (b) (c)

Figure 6.11: The effect of large-scale BoT applications for makespan M_c (number of VMs is 64). (a) 10% uncertain task proportion; (b) 20% uncertain task proportion; (c) 30% uncertain task proportion.



(a) (b) (c)

Figure 6.12: The effect of large-scale BoT applications on resource usage index ψ (number of VMs is 64). (a) 10% uncertain task proportion; (b) 20% uncertain task proportion; (c) 30% uncertain task proportion.

shows better M_c and ψ values. This is because when certain task queue is empty, uncertain tasks can use lesser number of VMs and hence a newly arrived *certain* task can obtain more resources to complete its execution earlier.

Further, one needs to determine the optimal number of VMs required to allocate for *uncertain* tasks, to minimize the values of M_c and ψ. This is evident from Fig. 6.10 and Fig. 6.12. We can notice that values of ψ is increased as more number of VMs are reserved for uncertain tasks or higher number of BoT applications are present in the system. So the CSPs need to fine-tune the values of various parameters based on the characteristics of the tasks coming to the system.

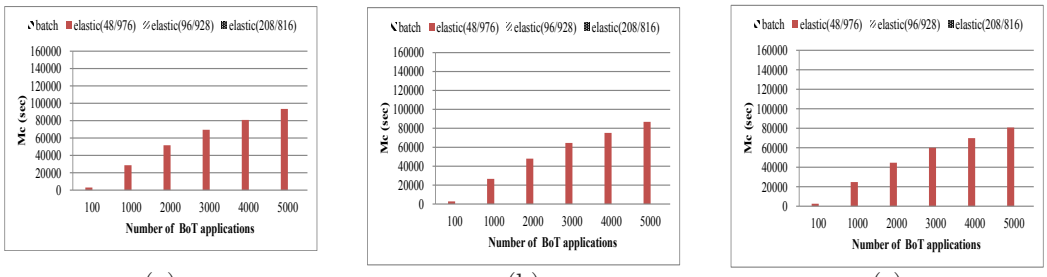
Figures 6.13 and 6.14 plot the value of M_c and ψ respectively when there are 1024 VMs in the virtual cluster. Having more VMs for the same number of applications results in better M_c and ψ in our scheme compared to the traditional scheme. For example, when there are 100 BoT applications, the shortest possible M_c and ψ values are obtained. But, note that this also means higher monetary cost for using larger number of VMs. This means that there is a trade-off between improving the value of M_c and lowering the monetary cost for using the VMs to

Table 6.10: Monetary costs (64 VMs).

Number of BoT applications	batch scheme	elastic scheme								
		10% uncertain task (l_1/l_0)			20% uncertain task (l_1/l_0)			30% uncertain task (l_1/l_0)		
		3/61 (a_1)	6/58 (a_2)	13/51 (a_3)	3/61 (a_4)	6/58 (a_5)	13/51 (a_6)	3/61 (a_7)	6/58 (a_8)	13/51 (a_9)
100	96	96	96	96	96	96	96	89.6	89.6	96
1000	921.6	857.6	870.4	889.6	857.6	857.6	876.8	800	806.4	870.4
2000	1760	1638.4	1708.8	1734.4	1587.2	1625.6	1670.4	1472	1612.8	1606.4
3000	2508.8	2323.2	2457.6	2464	2233.6	2304	2419.2	2150.4	2272	2329.6
4000	3187.2	2700.8	2867.2	3052.8	2598.4	2758.4	2892.8	2496	2649.6	2694.4
5000	3782.4	3276.8	3347.2	3475.2	3040	3110.4	3264	2700.8	2835.2	2886.4

run BoT applications.

Tables 6.10 and 6.11 summarize the monetary cost $Cost_{total}$ for all the BoT applications when the number of VMs are 64 and 1024 respectively. In this case, we use $c_k = 0.10$ (i.e. cost for resource/hour=0.10$/hour). It can be observed that, *elastic* scheme successfully reduces the overall monetary cost of the BoT

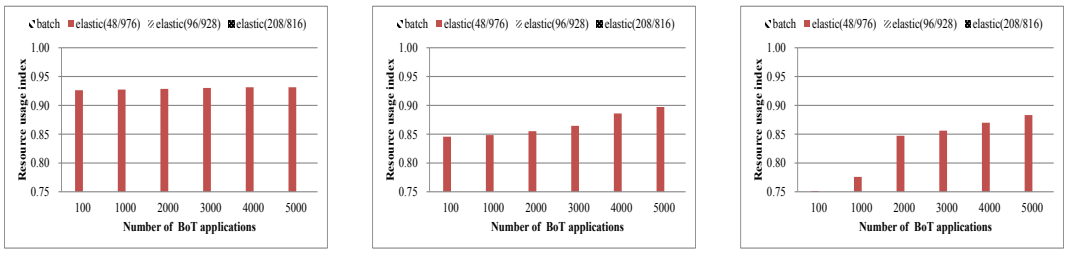


(a) (b) (c)

Figure 6.13: The effect of large-scale BoT applications for makespan M_c (number of VMs is 1024). (a) 10% uncertain task proportion; (b) 20% uncertain task proportion; (c) 30% uncertain task proportion.

Table 6.11: Monetary costs (1024 VMs).

Number of BoT applications	batch scheme	elastic scheme								
		10% uncertain task (l_1/l_0)			20% uncertain task (l_1/l_0)			30% uncertain task (l_1/l_0)		
		48/976 (b_1)	96/928 (b_2)	208/816 (b_3)	48/976 (b_4)	96/928 (b_5)	208/816 (b_6)	48/976 (b_7)	96/928 (b_8)	208/816 (b_9)
100	102.4	102.4	102.4	102.4	102.4	102.4	102.4	102.4	102.4	102.4
1000	921.6	819.2	921.6	921.6	819.2	819.2	921.6	716.8	819.2	819.2
2000	1843.2	1536	1536	1638.4	1433.6	1433.6	1536	1331.2	1433.6	1433.6
3000	2560	2048	2150.4	2150.4	1843.2	1945.6	2048	1740.8	1945.6	1843.2
4000	3276.8	2355.2	2457.6	2560	2150.4	2252.8	2355.2	2048	2150.4	2252.8
5000	4198.4	2662.4	2867.2	2969.6	2560	2662.4	2764.8	2355.2	2560	2560



(a) (b) (c)

Figure 6.14: The effect of large-scale BoT applications for resource usage index ψ (number of VMs is 1024). (a) 10% uncertain task proportion; (b) 20% uncertain task proportion; (c) 30% uncertain task proportion.

Table 6.12: Relative monetary costs of using 1024 VMs vs. 64 VMs

Number of BoT applications	Monetary cost of 64 VMs - Monetary cost of 1024 VMs								
	10% uncertain task (l_1/l_0)			20% uncertain task (l_1/l_0)			30% uncertain task (l_1/l_0)		
	a_1-b_1	a_2-b_2	a_3-b_3	a_4-b_4	a_5-b_5	a_6-b_6	a_7-b_7	a_8-b_8	a_9-b_9
100	-6.4	-6.4	-6.4	-6.4	-6.4	-6.4	-12.8	-12.8	-6.4
1000	38.4	-51.2	-32	38.4	38.4	-44.8	83.2	-12.8	51.2
2000	102.4	172.8	96	153.6	192	134.4	140.8	179.2	172.8
3000	275.2	307.2	313.6	390.4	358.4	371.2	409.6	326.4	486.4
4000	345.6	409.6	492.8	448	505.6	537.6	448	499.2	441.6
5000	614.4	480	505.6	480	448	499.2	345.6	275.2	326.4

applications under various situations by carefully scheduling the tasks based on priority. It can also be noted that, reserving more VMs (m_k) for *uncertain* tasks results in higher monetary cost. So one need to carefully choose the value of m_k based on his requirements.

We also compare the relative monetary cost for using 1024 VMs as compared to 64 VMs (Refer Table 6.12). When large number of BoT applications are present in the system, large number of resources helps in reducing the cost of execution. This also helps users to decide the number of VMs required to execute their tasks based on the application requirements and size.

6.5 Chapter Summary

In this chapter, we first proposed a model for aggregating BoT applications among multiple IaaS Compute CSPs with different resource capabilities. We modeled the CSPs as independent M/M/1 queues, formulated an optimization problem to minimize the task execution time and derived optimal solutions for task distribution. We further proposed a heuristic algorithm which considers not only the task execution time, but also the user's budget requirements in order to derive the task distribution.

We can observe that large scale data-intensive divisible load applications[1] such as image processing [89], [90] and biological computing [91], [92] applications, can also use our model for execution, to satisfy various constraints such as budget constraints and application execution time. In case of divisible load applications, we calculate the optimal amount of data to be distributed to each CSP instead of determining the number of tasks to be distributed. An exampel for data aggregation on Cloud environments is described in Appendix A. In addition to application execution time and budget constraints, users have many other considerations such as security, trust and reputation of CSP [93]. Using the *Operations Monitor* in our Broker, we can seamlessly integrate these features. Broker can consider a subset of all the CSPs satisfying such user criteria to derive the optimal task distribution.

Later, a strategy for the task scheduling of BoT applications within a Cloud has been proposed to minimize task execution time, resource usage and monetary cost. Tasks may be either a subset of tasks dispatched by the Broker or a set

[1] Divisible Load scheduling theory is referred to as Divisible Load Theory (DLT) in literature.

of BoTs submitted directly by users or both. These tasks are segregated in to two queues by assigning higher priority to *certain* tasks and lower priority to *uncertain* tasks. We model the Cloud as an M/M/m system and formulate the mathematical model to find out the makespan of the tasks in the system.

We conducted rigorous performance evaluation studies to evaluate our model and compared our scheme with traditional FCFS batch scheduling scheme. We specifically analyzed the influence on the amount of *uncertain* tasks present in the system, number of resources available for each task and the number of BoT applications present in the system and proved that our scheme considerably improves the makespan, reduces the resource usage and lowers the monetary cost incurred under all cases compared to batch scheme. One of the key findings from our studies is in demonstrating the trade-off relationship between improving makespan and lowering the monetary cost for using the resources to run BoT applications.

Chapter 7

Conclusions and Future Remarks

7.1 Conclusions

In this thesis, we have presented a Cloud-Broker architecture and Broker-mediated Cloud Orchestration mechanisms to connect users and CSPs through the Broker. Cloud users have several concerns for deploying their tasks/data into Cloud such as choosing the right CSP, security concerns, trustworthiness of CSPs etc. On the other hand, CSPs face several issues to enter and establish their business in this new distributed computing platform such as understanding and adapting to the market conditions, adapting to user requirements developing strategies to increase the revenue etc. In this thesis, we developed a comprehensive architecture for Cloud Broker and devised strategies for helping users and CSPs to make appropriate business decisions from time to time.

We have described a comprehensive survey on the state-of-the-art Cloud Broker mechanisms existing in the literature and categorized them into three based on the services offered by the Brokers. However the current Broker models are not capable of addressing various concerns by users as well as CSPs. From the

user perspective, Cloud Broker arbitrage strategies lack in efficient modeling of trust, reputation, security and user's risk bias. From the CSP's perspective, these Broker models lacks in efficient mechanisms to understand and adapt to market conditions in terms of devising market-oriented dynamic pricing strategies and modeling user reliability.

In the first part of this thesis, we proposed three Cloud Broker arbitrage mechanisms to address the above issues based on incentives, auction theory and VNM utility theory. We proposed dynamic pricing strategies for CSPs based on the market conditions, non-cooperative repeated game theoretic model for measuring the reliability of users and mathematical formulations for calculating and maintaining various parameters such as trust, reputation etc. We have conducted extensive performance evaluation studies to understand the effect of various system parameters under different market conditions. We found that incentive based scheme is suitable for loyal users who use the Cloud for long time whereas sealed-bid continuous double auction based model is suitable for users who use Cloud in an ad-hoc manner. The scheme based on VNM utility theory is particularly useful for users who are risk-averse in nature.

In the second part of this thesis, we proposed two Cloud Broker aggregation mechanisms for IaaS Clouds one based on cooperative games and the other one based on Markovian queues. We employed bargaining solutions propounded in literature to efficiently determine the resource requirements for a set of tasks requesting for one type of resources so as to maximize the resource utilization and to handle elastic user requirements. An important contribution of this work is in introducing an asymmetric pricing scheme which takes into account user's budget

requirements for resource allocation.

Further, we proposed a Cloud Broker aggregation mechanism for deploying tasks in BoT applications and applications following divisible load theory, among multiple Clouds with different resource capabilities. We modeled the problem using Markovian queues and formulated an optimization problem. We considered different user requirements such as deadline and budget specifications for determining the task aggregation. We further modeled task aggregation among a single CSP with different datacenters using queueing theory and discussed efficient task scheduling strategies to minimize makespan and improve resource usage. We compared the results with existing strategies and showed that our scheduling mechanism is effective under different situations.

Thus, our Broker can incorporate several features suitable for various situations. It enables different arbitrage mechanisms for various kinds of users such as loyal customers, risk-averse users etc. It also incorporates several intermediation features such as trust, reputation and security indices about CSPs for users. Further, the Broker enables aggregation of a set of tasks/data among multiple CSPs to satisfy certain user requirements such as budget and/or deadline. Our Broker can function either as an entity to connect several CSPs and users or as an entity to connect several users to one CSP. It also helps CSPs to monitor the market and develop market-oriented pricing strategies and understand user behavior in order to adapt to the market situations.

7.2 Future Work

The Broker architecture discussed in this thesis has solved various issues pertaining to users and CSPs in using Cloud environments efficiently. However, there are still many issues left to be addressed. In this section, we briefly discuss some of these issues.

1. The Broker arbitrage mechanisms described in first part of the thesis can effectively handle several issues such as trust, risk etc. However, *vendor lock-in* is not discussed in this thesis. In order to handle vendor lock-in issues, we need to know the exact profile of all the CSPs participating in the market and efficient mathematical models are required to form the *switching cost model*. The switching cost includes possible end-of-contract penalties, charges for format conversion and data/application switching and possible additional charges for bandwidth usage.

2. The Broker can create a *spot market* where the CSPs can participate in an auction market to offer their unused resources temporarily for short duration of time. This could be achieved by creating a market place wherein CSPs can inform Brokers about the amount and type of resources available and users can bid for such limited resources. Special mathematical models are required to formulate the spot pricing strategies based on supply and demand of resources in a competitive environment.

3. The Broker architecture can also be extended to support few Broker intermediation services such as *Singel Sign-On (SSO)*. SSO can help users to log-in only once to access all Cloud services available through the Broker.

It will also help users to access unified billing information and payment options. A SSO mechanism allows users to change their security information such as password and billing details in a single place (at the Broker) for all providers.

Appendix: Example for Data Aggregation on Cloud - Large-scale Polynomial Multiplication

A.1 Introduction

As mentioned in Chapter 6, we can deploy large-scale divisible load applications among multiple resources (belonging to one or more CSPs) to minimize the application execution time. Now, we describe the deterministic modeling of one such application considering various system parameters. Users can submit such applications to Broker and Broker can determine the fraction of data to be distributed to different resources based on different system parameters and the amount of input data.

Large scale polynomial product computations often used in applications such as image processing and sensor networks data processing always pose consider-

able challenge when processed on networked computing systems. With non-zero communication and computation time delays of the links and processors, the computation becomes all the more challenging. We use Divisible Load Theory (DLT) [94] to design efficient strategies to minimize the overall processing time for performing large scale polynomial product computations in Cloud environments. We consider a Cloud system (It can also be a set of CSPs among which the data is distributed) with the Broker distributing the entire load to a set of virtual CPU instances (VCI) and the VCIs propagating back the processed results to Broker for post-processing. Thus Broker offers intermediation features such as determining the optimal number of resources required and final post-processing of the results in addition to data aggregation.

We use Point Value Representation (PVR) method for polynomial product computation. It primarily involves four phases - *Selection*, *Evaluation*, *Multiplication*, and *Interpolation* respectively [90]. With very large degree-bound n for the polynomials, the time taken for product computations become prohibitively large and hence, inputs to the VCIs are the point values selected in the Selection phase. Also, it may be noted that the product computations are independent and hence they can be divided arbitrarily among different VCIs.

As the problem setting demands a post-processing phase, we also consider *solution back propagation* in our modeling. We consider only the case where each *main VCI* for computation has an associated *front-end VCI* for the communication and hence a pair of VCIs can perform both computation (by main VCI) and communication (by front-end VCI) operations concurrently. We also assume that the polynomials to be used are available at the Broker. For simplicity, we

denote the term *main VCI* as *processor* in the remainder of the paper. We shall now define the following terms:

1.Load Distribution: It is denoted as α and is defined as an m-tuple (α_1, . . . ,α_m) such that $0 \leq \alpha_i \leq m$ and $\sum_{i=0}^{m} \alpha_i = 1$. This is called as a normalization equation.

2. Processing Time: This is the total computation time taken by the resource allocator VCI for a *Compute Cloud* system.

3. Availability of Link (Link availability) - The term availability of a link is defined as the percentage of the link capacity that is available to transfer the load from the resource allocator VCI to other processors.

4. Availability of VCI (VCI availability) - The term availability of a VCI is defined as the percentage of the VCI capacity that is available for computation of the load fractions assigned to that VCI.

5. Optimal Sequence: This is defined as the sequence of optimal load distribution for a given arrangement such that the processing time is minimum.

6. Optimal Sequence Theorem (OST): In order to achieve minimum processing time in a single-level tree network, the sequence of load distribution by the root processor p_0 should follow the order in which the link speed decrease.

A.2 Analysis For the Load Fractions

The Cloud consists of $m + 1$ *processors*/VCIs ($p_0, p_1, p_2, ..., p_m$) with p_0 as the Broker and m communication links ($l_1, l_2, ...l_m$). Each of these VCIs has an associated *front-end VCI* for the communication of the load (The *aggregation* module in case of Broker). The VCIs and links are considered to be heterogeneous i.e.,

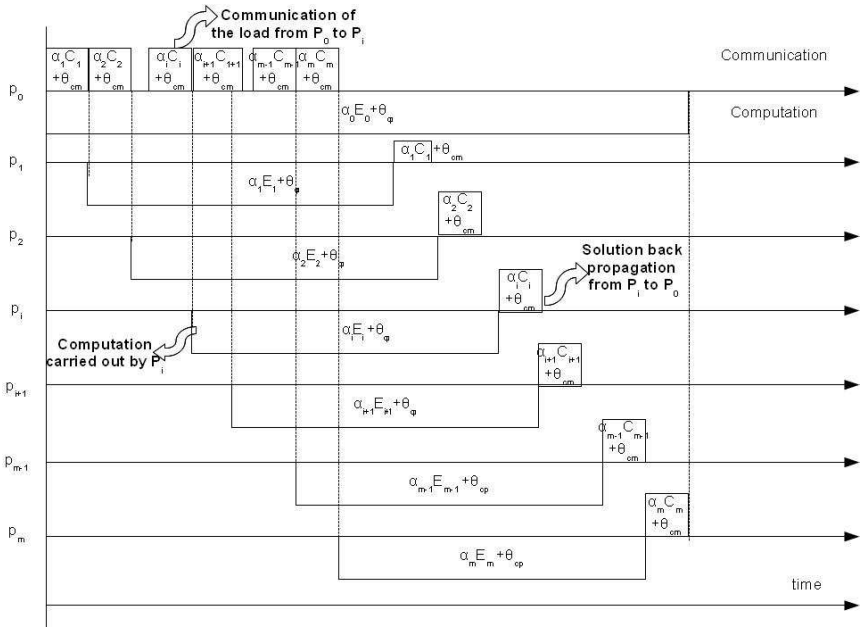


Figure A.1: Timing Diagram for Compute Cloud with solution-back propagation

they have different processing and communication speeds, respectively. The load for the *processors* are the $2n$ points (for the *Evaluation* phase) which are selected by the Broker p_0. Initially, it is assumed that the load is available/submitted at p_0. The availabilities of the links and the *processors*, denoted as A_i^l and A_i^p. respectively, are also considered. We assume that the Broker p_0 is always available, i.e., A_0^p=1.

We now derive a closed form solution for the optimal load fraction for each processor. It is assumed that the load distribution is from p_1 to p_m. The fraction of the total load assigned to a processor p_i is denoted as α_i. The process of load distribution is represented in the form of a timing diagram shown in Figure A.1 as described in the literature [94]. From the timing diagram, we obtain the following

recursive equations for the load fractions to the processors[1].

$$\frac{\alpha_i E_i}{A_i^p} + \theta_{cp} = \frac{\alpha_{i+1} C_{i+1}}{A_{i+1}^l A_{i+1}^p} + \frac{\alpha_{i+1} E_{i+1}}{A_{i+1}^p} + \theta_{cm} + \theta_{cp} - (\frac{\alpha_i C_i}{A_i^l A_i^p} + \theta_{cm}) \tag{A.1}$$

for $i = 1, 2, ..., m-1$.

It can be noted that, the overheads will have their influence only when a link and/or a processor, is available and largely comes as additive parameters [95] (as followed and demonstrated in DLT literature so far).

We can express α_i as:

$$\alpha_i = \frac{\alpha_{i+1} a_{i+1}}{a_i} \tag{A.2}$$

where $a_i = \frac{E_i}{A_i^p} + \frac{C_i}{A_i^l A_i^p}$. Expressing each of the load fractions in terms of α_m, we obtain,

$$\alpha_i = \frac{\alpha_m M_i}{N_i} \tag{A.3}$$

where $M_i = \prod_{j=i+1}^{m} a_j$ and $N_i = \prod_{g=i}^{m-1} a_g$. From Figure A.1, we obtain the total processing time as follows.

$$\alpha_0 E_0 + \theta_{cp} = \sum_{i=1}^{m} (\frac{\alpha_i C_i}{A_i^l A_i^p} + \theta_{cm}) + \frac{\alpha_m E_m}{A_m^p} + \theta_{cp} + \frac{\alpha_m C_m}{A_m^l A_m^p} + \theta_{cm} \tag{A.4}$$

$$\alpha_0 = \alpha_1 d_1 + \alpha_2 d_2 + ... + \alpha_m d_m + \alpha_m h + \alpha_m d_m + \frac{(m+1)\theta_{cm}}{E_0} \tag{A.5}$$

[1]In this definition, without loss of generality, we assume $\theta_{cm}^{(i)} = \theta_{cm}$ and $\theta_{cp}^{(i)} = \theta_{cp} \forall i$ for the sake of analytical ease

where $d_i = \frac{C_i/A_i^l A_i^p}{E_0}$ and $h = \frac{E_m/A_m^p}{E_0}$ Now we will derive an expression for obtaining the load fraction α_m assigned to the mth processor. The normalization equation is given by $\sum_{i=0}^{m} \alpha_i = 1$

Expanding the normalization equation and substituting the value of α_0 from (A.5), we have,

$$\alpha_1(1+d_1) + \alpha_2(1+d_2) + ... + \alpha_m(1+d_m+h+d_m) = 1 - \frac{(m+1)\theta_{cm}}{E_0} \quad \text{(A.6)}$$

Expressing each α_i in terms of α_m from (A.3) and further simplifying we obtain α_m as,

$$\alpha_m = \frac{1 - \frac{(m+1)\theta_{cm}}{E_0}}{(1 + 2d_m + h + \sum_{i=1}^{m-1}(\frac{(1+d_i)(\prod_{j=i+1}^{m} a_j)}{\prod_{g=1}^{m-1} a_g}))} \quad \text{(A.7)}$$

Let

$$X_m = \frac{(m+1)\theta_{cm}}{E_0} \quad \text{(A.8)}$$

and

$$Y_m = (1 + 2d_m + h + \sum_{i=1}^{m-1}(\frac{(1+d_i)(\prod_{j=i+1}^{m} a_j)}{\prod_{g=1}^{m-1} a_g})) \quad \text{(A.9)}$$

then (A.7) can be rewritten as,

$$\alpha_m = \frac{1 - X(m)}{Y(m)} \quad \text{(A.10)}$$

From the expression for overhead factor (A.8), we can obviously infer that as the number of processors (m) increases, the overhead factor also increases and this overhead factor is also directly proportional to θ_{cm}. For a given load size,

we may not require all $(m+1)$ available processors (and hence the front-end VCIs). The maximum number of processors that can be used for a given load size can be calculated easily by knowing the overhead factor. We can see that when $X(m)$ is greater than or equal to 1, then the load fraction for processor m becomes negative i.e., the overhead factor dominates. So a necessary and sufficient condition to obtain a maximal number of processors that can be used for a given load size is given by,

$$X(m) = \frac{(m+1)\theta_{cm}}{E_0} < 1 \tag{A.11}$$

Thus in a given network of sufficiently large number of VCIs, say M, we need to choose only $k^* \leq M$ processors that need to be assigned load fractions, where k^* satisfies (A.11). This maximal set of processors k^* can be determined recursively using the recursive equations derived above. Obviously this consumes $O(m)$ time. We also verify these findings via our simulation experiments.

A.3 Performance Evaluation and Discussions of the Results

In our simulation experiments, we generate random values for processor speeds, link speeds, and processor and link availabilities. We use two polynomials of degree $10,000$ each for our simulation experiments. The coefficients of all the points are randomly generated following a uniform distribution at the resource allocator p_0. Then we derive the load fractions to be assigned to each processor available in the Cloud and then we apply integer approximation technique. Polynomials are

Table A.1: Simulation parameters

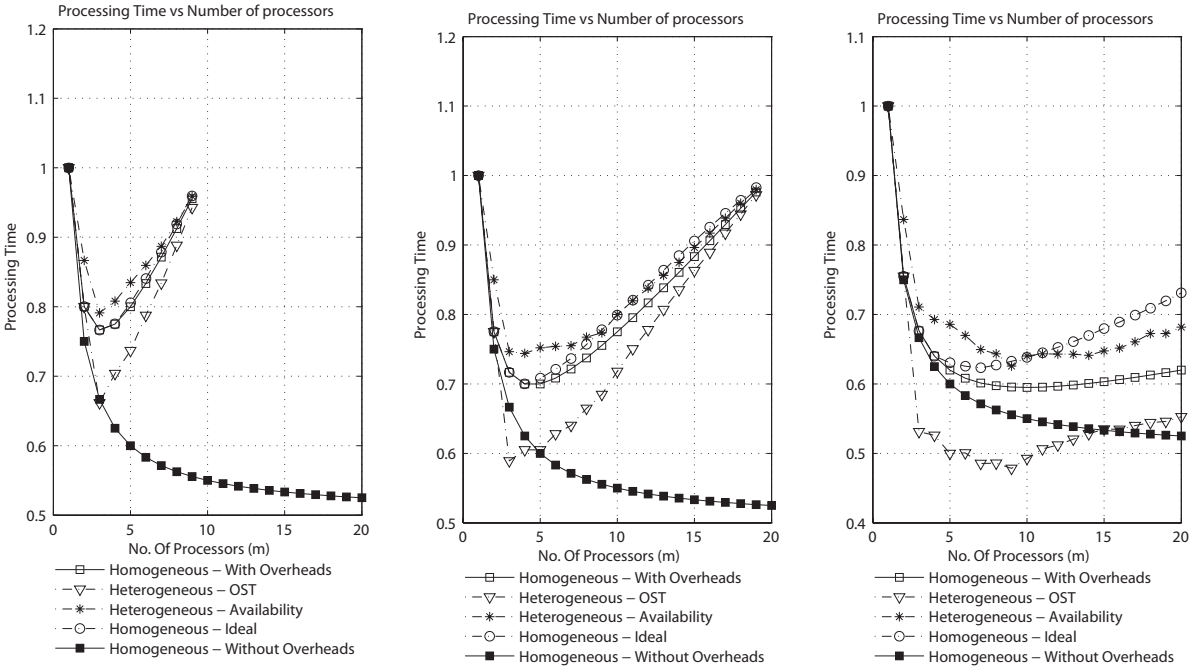
L	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8	C_9	C_{10}	C_{11}	C_{12}	C_{13}	C_{14}	C_{15}	C_{16}	C_{17}	C_{18}	C_{19}
20,000	1	0.2	0.9	0.6	0.8	0.5	0.7	0.4	1.2	1.3	0.8	1	0.9	0.9	0.5	0.8	0.7	0.6	0.9
E_0	E_1	E_2	E_3	E_4	E_5	E_6	E_7	E_8	E_9	E_{10}	E_{11}	E_{12}	E_{13}	E_{14}	E_{15}	E_{16}	E_{17}	E_{18}	E_{19}
0.9	1	0.7	0.9	0.5	0.8	0.6	0.4	1	1.2	1.2	0.9	1.1	0.3	0.6	0.9	0.7	0.8	0.5	1.1
A_0^p	A_1^p	A_2^p	A_3^p	A_4^p	A_5^p	A_6^p	A_7^p	A_8^p	A_9^p	A_{10}^p	A_{11}^p	A_{12}^p	A_{13}^p	A_{14}^p	A_{15}^p	A_{16}^p	A_{17}^p	A_{18}^p	A_{19}^p
1	0.7	0.4	0.9	0.5	0.8	0.6	0.7	0.85	1	1	0.7	0.9	1	0.8	0.4	0.6	0.5	0.9	0.6
A_0^l	A_1^l	A_2^l	A_3^l	A_4^l	A_5^l	A_6^l	A_7^l	A_8^l	A_9^l	A_{10}^l	A_{11}^l	A_{12}^l	A_{13}^l	A_{14}^l	A_{15}^l	A_{16}^l	A_{17}^l	A_{18}^l	A_{19}^l
–	0.8	1	0.6	0.7	0.8	0.9	0.8	0.75	0.5	0.9	1	1	0.8	0.7	0.9	0.6	0.4	0.6	0.5

evaluated using the PVR method and multiplied and the resulting polynomial is obtained using interpolation as described earlier. Values of different parameters used for simulation are given in Table A.1[1]. We carried out the simulation experiments for polynomial multiplication for the following cases:

- Homogeneous networks with and without overheads (i.e. θ_{cp} and θ_{cm}) with $A_i^l = A_j^p = 1,\ \forall i, j$
- Homogeneous networks in an ideal case where we consider the scenario with highly available links and VCIs are connected together (In the other cases, highly available links may not be connected to highly available VCIs.)
- Heterogeneous networks using Optimal Sequence Theorem (OST)
- Heterogeneous networks with processor and link availability

We consider processing time as performance metric. It is studied for different values of communication overheads θ_{cm}, as it has direct influence in deciding a maximal set of VCIs to be used and hence on the performance too. This also influences the number of VCIs needs to be requested from the Cloud Service Provider and determines the economic factors.

[1]Note that the values of E_i and C_i are set to 1 for homogeneous experiments.



(a) (b) (c)

Figure A.2: Processing Time vs Number of processors (a) $\theta_{cm} = 0.01$ (b)$\theta_{cm} = 0.05$ (c) $\theta_{cm} = 0.1$

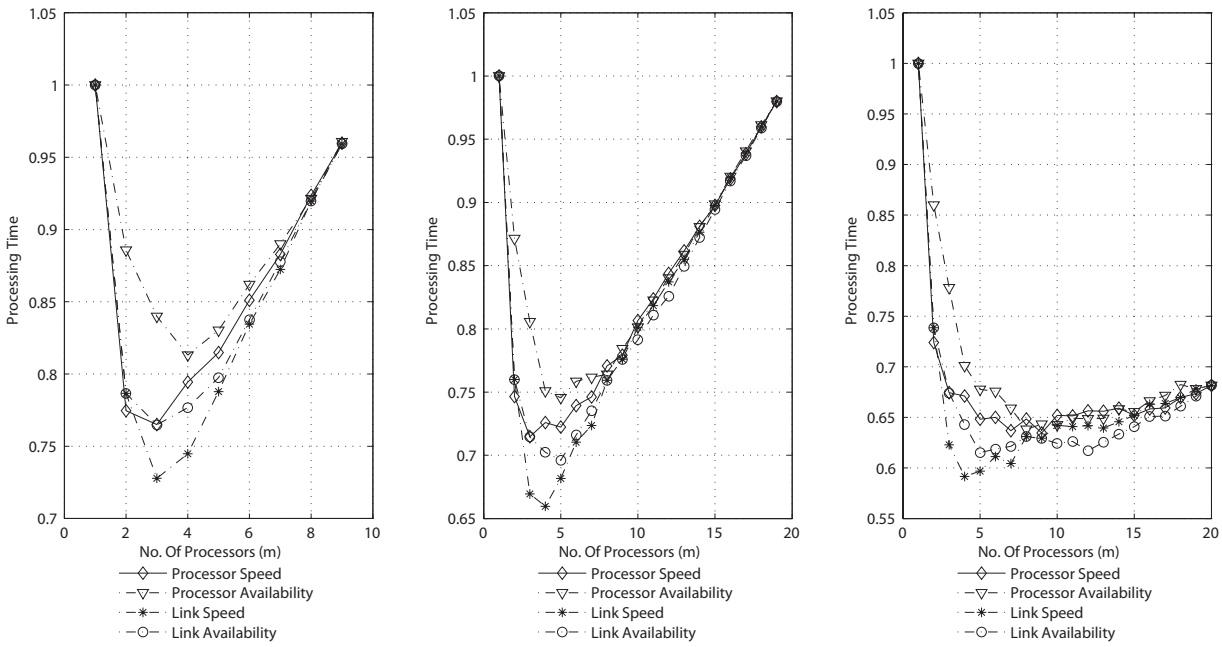
A.3.1 Processing time

The performance of the system in terms of processing time with respect to number of processors for different values of θ_{cm} are plotted in Figures 2(a), 2(b) and 2(c). When a homogeneous *Compute Cloud* with no overheads is considered, the processing time decreases with the increase in the number of processors. But for all other cases, the processing time decreases initially with more number of processors up to some extent and then the processing time increases if we increase the number of processors. This is due to the effect of $X(m)$ in equation (A.10). That is, as the communication overhead increases, the system takes more time to transfer the load to leaf nodes. As the value of θ_{cm} increases, $X(m)$ increases

and hence α_m, the load to processor m decreases. From equation (A.3), it is clear that as the value of α_m decreases, the load fractions for leaf nodes α_iś also decreases linearly. This leads to an increased load in resource allocator and hence the processing time increases. This effect is more evident in Figures 2(b) and 2(c) for the cases with and without overheads.

For example, when the value of θ_{cm} is 0.01, it uses all processors and the processing time is least when the number of processors is $m = 9$ in case of heterogeneous *Compute Cloud* system, between 8-12 for homogeneous *Compute Cloud* with overheads and $m = 6$ for homogeneous *Compute Cloud* under an ideal condition. The ideal condition is that, we physically connect highly available links to highly available processors in a decreasing order and distribute the load. In case of homogeneous *Compute Cloud* with no overheads, we can observe that beyond $m = 12$, the rate at which the processing time decreases is minimum. Hence one can utilize 12 processors without having to utilize all 20 processors to achieve high performance.

When the value of θ_{cm} is 0.05, the maximum number of processors required to process the entire load is less than the number of available processors in some cases. For example, the maximum number of processors required to process the entire load in case of a heterogeneous *Compute Cloud* using OST is 19 even though 20 processors are available to do the operation. This is because, if the number of processors are increased beyond this value, then the overhead outweighs the processing time. Also the processing time is minimum when the number of processors is 3 for a heterogeneous *Compute Cloud* using OST and 4 for all other cases.



(a) (b) (c)

Figure A.3: Processing Time: Influence of system characteristics in selecting the required VCIs when the number of processors required is less than the available number of processors (a) $\theta_{cm} = 0.01$ (b)$\theta_{cm} = 0.05$ (c) $\theta_{cm} = 0.1$

When we further increase the value of θ_{cm} to 0.1, then the optimal number of processors is further reduced. In all cases except the case when the overhead is not considered for a homogeneous *Compute Cloud*, the optimal number of processors required to process the entire load is only 3.

A.3.2 Strategies for eliminating redundant processors

In all above cases, when the number of processors required is less than the actual number of available processors, we selected the required number of processors in sequence. In the Figures 3(a), 3(b) and 3(c), we show the influence of system characteristics - link speed, processor speed, link availability and processor availability - in selecting the required processors when the number of processors

required is less than the available number of processors. The figures contain plots for four different cases - (1) Eliminate the slowest links; (2) Eliminate the slowest processors; (3) Eliminate the least available links and 4) Eliminate the least available processors.

Processing time is the least in case when we eliminate the slower links for all values of θ_{cm}. The optimal processing time obtained by eliminating slower links is less compared to that obtained by eliminating slower processors. This is because a faster processor can have a slow (or poorly available) link which will take more time to communicate. Similarly, the optimal processing time obtained by selecting processors with higher link availability compared to the processing time obtained by selecting processors with higher processor availability.

A.4 Summary

Thus we can observe that for a given load and application, Broker can derive optimal number of processors required for processing as well as the fraction of data for each processor based on network heterogeneity. In this way, Broker can incorporate some important intermediation services which can help users to reduce their application execution time and the total monetary cost. Further, the application described here is a representative example and any user application which comes under DLT and needs a post processing can use this model to derive the load fractions and optimal number of resources required.

References

[1] Jackie Fenn, Pete Basiliere, Kathy Harris, and Daryl C. Plummer. After the next big thing: The consequences of a cloud computing scenario. Technical report, Gartner Inc., May 2009. xiii, 12

[2] Peter R. Wurman, Michael P. Wellman, and William E. Walsh. A parametrization of the auction design space. *Games and Economic Behavior*, 35(1-2):304–338, April 2001. xiii, 35

[3] Kai Hwang, Geoffrey Fox, and Jack Dongarra. *Distributed and Cloud Computing: From Parallel Processing to the Internet of Things*. Morgan Kaufmann, 10/2011 2011. xix, 2, 3

[4] Buyya Rajkumar, Yeo Chee Shin, Venugopal Srikumar, Broberg James, and Brandic Ivona. Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Gener. Comput. Syst.*, 25(6):599–616, June 2009. xix, 3

[5] Cloudorado. Cloud computing price comparison engine, March 2012. http://www.cloudorado.com/. xix, 62, 63

[6] Armbrust Michael, Fox Armando, Griffith Rean, Joseph Anthony D.,

Katz Randy H., Konwinski Andrew, Lee Gunho, Patterson David A., Rabkin Ariel, Stoica Ion, and Zaharia Matei. Above the clouds: A berkeley view of cloud computing. Technical Report UCB/EECS-2009-28, EECS Department, University of California, Berkeley, Feb 2009. 1

[7] Mell; Peter and Timothy Grance. The nist definition of cloud computing, 2011. 1

[8] Mdric Morel, Manuel Alves, and Pascal Cadet. *Google Apps: Mastering Integration and Customization*. Packt Publishing, Limited, 2011. 2

[9] Sitaram D. and Manjunath G. *Moving To The Cloud: Developing Apps in the New World of Cloud Computing*. Elsevier Science, 2011. 4

[10] Charles Severance. *Using Google App Engine*. O'Reilly Series. O'Reilly, 2009. 4

[11] Tom Rizzo, Razi bin Rais, Michiel van Otegem, Darrin Bishop, George Durzi, Zoiner Tejada, and David Mann. *Programming Microsoft's Clouds: Azure and Office 365*. John Wiley and Sons, 2012. 4

[12] Murty J. *Programming Amazon Web Services: S3, EC2, SQS, FPS, and SimpleDB*. O'Reilly Series. O'Reilly, 2008. 4, 49, 103

[13] J. Van Vliet and F. Paganelli. *Programming Amazon Ec2*. Head First Series. O'Reilly Media, 2011. 6

[14] David W. Cearley and David Mitchell Smith. Five cloud computing trends that will affect your cloud strategy through 2015. Technical report, Gartner Inc., Feb 2012. 12

[15] Attila Kertesz and Peter Kacsuk. A taxonomy of grid resource brokers. In Pter Kacsuk, Thomas Fahringer, and Zsolt Nmeth, editors, *Distributed and Parallel Systems*, pages 201–210. Springer US, 2007. 13

[16] Rajkumar Buyya, David Abramson, Jonathan Giddy, and Heinz Stockinger. Economic models for resource management and scheduling in grid computing. *Concurrency and Computation: Practice and Experience*, 14(13-15):1507–1542, 2002. 13

[17] A. Kertesz and T. Prokosch. The anatomy of grid resource management. In Franco Davoli, Norbert Meyer, Roberto Pugliese, and Sandro Zappatore, editors, *Remote Instrumentation and Virtual Laboratories*, pages 123–132. Springer US, 2010. 13

[18] Bo An, Victor Lesser, David Irwin, and Michael Zink. Automated negotiation with decommitment for dynamic resource allocation in cloud computing. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1 - Volume 1*, AAMAS '10, pages 981–988, Richland, SC, 2010. International Foundation for Autonomous Agents and Multiagent Systems. 14, 25

[19] Mohan Baruwal Chhetri, Quoc Bao Vo, Ryszard Kowalczyk, and Cam Lan Do. Cloud broker: helping you buy better. In *Proceedings of the 12th international conference on Web information system engineering*, WISE-2011, pages 341–342, Berlin, Heidelberg, 2011. Springer-Verlag. 14, 25

[20] Y Sundeep, Saurabh Kumar Jain, Saranya H V, Sai Manoj P D, and Kailash Chandra Raythour. Cloud broker. Technical Report IIITB-OS-2011-

5D, International Institute of Information Technology, Bangalore, April 2011. 14, 25

[21] Venkatarami Reddy Chintapalli. A deadline and budget constrained cost and time optimization algorithm for cloud computing. In Ajith Abraham, Jaime Lloret Mauri, John F. Buford, Junichi Suzuki, and Sabu M. Thampi, editors, *Advances in Computing and Communications*, volume 193 of *Communications in Computer and Information Science*, pages 455–462. Springer Berlin Heidelberg, 2011. 14, 25

[22] Fei Teng and Frederic Magoules. A new game theoretical resource allocation algorithm for cloud computing. In Paolo Bellavista, Ruay-Shiung Chang, Han-Chieh Chao, Shin-Feng Lin, and Peter Sloot, editors, *Advances in Grid and Pervasive Computing*, volume 6104 of *Lecture Notes in Computer Science*, pages 321–330. Springer Berlin, Heidelberg, 2010. 14, 25

[23] Kassidy P. Clark, Martijn Warnier, and Frances M. T. Brazier. An intelligent cloud resource allocation service - agent-based automated cloud resource allocation using micro-agreements. In *In the proceedings of the 2nd International Conference on Cloud Computing and Services Science (CLOSER 2012)*, page ??, 2012. 15, 25

[24] Saurabh Kumar Garg, Christian Vecchiola, and Rajkumar Buyya. Mandi: a market exchange for trading utility and cloud computing services. *Accepted for publication in The Journal of Supercomputing*, 2011. 15, 25

[25] Mohsen Amini Salehi and Rajkumar Buyya. Adapting market-oriented scheduling policies for cloud computing. In *Proceedings of the 10th inter-*

national conference on Algorithms and Architectures for Parallel Processing - Volume Part I, ICA3PP'10, pages 351–362, Berlin, Heidelberg, 2010. Springer-Verlag. 15, 25

[26] Praveen Ganghishetti, Rajeev Wankar, Rafah M. Almuttairi, and C. Raghavendra Rao. Rough set based quality of service design for service provisioning in clouds. In *Proceedings of the 6th international conference on Rough sets and knowledge technology*, RSKT'11, pages 268–273, Berlin, Heidelberg, 2011. Springer-Verlag. 16, 25

[27] Rodrigo N. Calheiros, Adel Nadjaran Toosi, Christian Vecchiola, and Rajkumar Buyya. A coordinator for scaling elastic applications across multiple clouds. *Future Generation Computer Systems*, (0):–, 2012. 16, 25

[28] Shifeng Shang, Jinlei Jiang, Yongwei Wu, Guangwen Yang, and Weimin Zheng. A knowledge-based continuous double auction model for cloud market. In *Proceedings of the 2010 Sixth International Conference on Semantics, Knowledge and Grids*, SKG '10, pages 129–134, Washington, DC, USA, 2010. IEEE Computer Society. 16, 25

[29] Salvatore Venticinque, Rocco Aversa, Beniamino Di Martino, Massimilano Rak, and Dana Petcu. A cloud agency for sla negotiation and management. In *Proceedings of the 2010 conference on Parallel processing*, Euro-Par 2010, pages 587–594, Berlin, Heidelberg, 2011. Springer-Verlag. 14, 25

[30] Mohammad Mehedi Hassan and Eui-Nam Huh. Resource management for data intensive clouds through dynamic federation: A game theoretic ap-

proach. In Borko Furht and Armando Escalante, editors, *Handbook of Data Intensive Computing*, pages 169–188. Springer New York, 2011. 17, 26

[31] Pradeep Kumar Tripathi, Surendra Mishra, and Pankaj Kawadkar. Cloud aggregation and bursting for object based sharable environment. *International Journal of Advanced Computer Research (IJACR)*, 1(3):5, 2011. 17, 26

[32] Ines Houidi, Marouen Mechtri, Wajdi Louati, and Djamal Zeghlache. Cloud service delivery across multiple cloud platforms. In *Proceedings of the 2011 IEEE International Conference on Services Computing*, SCC '11, pages 741–742, Washington, DC, USA, 2011. IEEE Computer Society. 17, 26

[33] Eric Kuada and Henning Olesen. A social network approach to provisioning and management of cloud computing services for enterprises. In *The Second International Conference on Cloud Computing, GRIDs, and VirtualizationThe Second International Conference on Cloud Computing, GRIDs, and Virtualization, Rome, Italy*, CLOUD COMPUTING 2011, pages 98–104, 2011. 17, 26

[34] Alistair Barros and Uwe Kylau. Service delivery framework - an architectural strategy for next-generation service delivery in business network. In *Proceedings of the 2011 Annual SRII Global Conference*, SRII '11, pages 47–58, Washington, DC, USA, 2011. IEEE Computer Society. 17, 26

[35] J. Gutierrez-Garcia and Kwang Sim. Ga-based cloud resource estimation for agent-based execution of bag-of-tasks applications. *Information Systems Frontiers*, pages 1–27. 10.1007/s10796-011-9327-8. 17, 26

[36] Xiaoyu Yang, Bassem Nasser, Mike Surridge, and Stuart Middleton. A business-oriented cloud federation model for real-time applications. *Future Generation Computer Systems*, (0):–, 2012. 18, 26

[37] Johan Tordsson, Rubn S. Montero, Rafael Moreno-Vozmediano, and Ignacio M. Llorente. Cloud brokering mechanisms for optimized placement of virtual machines across multiple providers. *Future Generation Computer Systems*, 28(2):358 – 367, 2012. 18, 26

[38] Ruben Van den Bossche, Kurt Vanmechelen, and Jan Broeckhove. Cost-optimal scheduling in hybrid iaas clouds for deadline constrained workloads. In *Proceedings of the 2010 IEEE 3rd International Conference on Cloud Computing*, CLOUD '10, pages 228–235, Washington, DC, USA, 2010. IEEE Computer Society. 18, 26

[39] Jose Luis Lucas-Simarro, Rafael Moreno-Vozmediano, Ruben S. Montero, and Ignacio M. Llorente. Scheduling strategies for optimal service deployment across multiple clouds. *Future Generation Computer Systems*, (0):–, 2012. 18, 26

[40] Americo Sampaio and Nabor Mendonca. Uni4cloud: an approach based on open standards for deployment and management of multi-cloud applications. In *Proceedings of the 2nd International Workshop on Software Engineering for Cloud Computing*, SECLOUD '11, pages 15–21, New York, NY, USA, 2011. ACM. 18, 26

[41] OCCI. Open cloud computing interface (occi), 2012. http://occi-wg.org/. 18

[42] OVF. Open virtualization format (ovf), 2012. http://www.dmtf.org/standards/ovf. 18

[43] Srijith K. Nair, Sakshi Porwal, Theo Dimitrakos, Ana Juan Ferrer, Johan Tordsson, Tabassum Sharif, Craig Sheridan, Muttukrishnan Rajarajan, and Afnan Ullah Khan. Towards secure cloud bursting, brokerage and aggregation. In *Proceedings of the 2010 Eighth IEEE European Conference on Web Services*, ECOWS '10, pages 189–196, Washington, DC, USA, 2010. IEEE Computer Society. 19, 26

[44] Kong E. Cheng, Yitzchak M. Gottlieb, Gary M. Levin, and Fuchun Joe Lin. Service brokering and mediation: Enabling next generation market and customer driven service delivery. In *Proceedings of the 2011 Tenth International Symposium on Autonomous Decentralized Systems*, ISADS '11, pages 525–530, Washington, DC, USA, 2011. IEEE Computer Society. 19, 27

[45] Theo Dimitrakos. Common capabilities for service oriented infrastructures and platforms: An overview. In *Proceedings of the 2010 Eighth IEEE European Conference on Web Services*, ECOWS '10, pages 181–188, Washington, DC, USA, 2010. IEEE Computer Society. 19, 27

[46] Stella Gatziu Grivas, Tripathi Uttam Kumar, and Holger Wache. Cloud broker: Bringing intelligence into the cloud. In *Proceedings of the 2010 IEEE 3rd International Conference on Cloud Computing*, CLOUD '10, pages 544–545, Washington, DC, USA, 2010. IEEE Computer Society. 20, 27

[47] Huaglory Tianfield. Cloud computing architectures. In *Proceedings of the*

IEEE International Conference on Systems, Man and Cybernetics, Anchorage, Alaska, USA, October 9-12, pages 1394–1399, 2011. 20, 27

[48] Zhiyun Guo, Meina Song, and Qian Wang. Policy-based market-oriented cloud service management architecture. In Yanwen Wu, editor, *Computing and Intelligent Systems*, volume 233 of *Communications in Computer and Information Science*, pages 284–291. Springer Berlin Heidelberg, 2011. 20, 27

[49] Gaurav Raj. An efficient broker cloud management system. In *Proceedings of the International Conference on Advances in Computing and Artificial Intelligence*, ACAI '11, pages 72–76, New York, NY, USA, 2011. ACM. 20, 27

[50] Lee Gillam, Bin Li, and John O.Loughlin. Adding cloud performance to service level agreements. 2nd International Conference on Cloud Computing and Services Science, CLOSER 2012, 2011. 20, 27

[51] He Yuan Huang, Bin Wang, Xiao Xi Liu, and Jing Min Xu. Identity federation broker for service cloud. In *Proceedings of the 2010 International Conference on Service Sciences*, ICSS '10, pages 115–120, Washington, DC, USA, 2010. IEEE Computer Society. 20, 27

[52] C. Pahl, V. Gacitua-Decar, M.X. Wang, and K.Y. Bandara. Flexible coordination techniques for dynamic cloud service collaboration. In Guadalupe Ortiz and Javier Cubo, editors, *Adaptive Web Services for Modular and Reusable Software Development: Tactics and Solution*, page 411. IGI Global, 2012. 20, 27

[53] Yichao Yang, Yanbo Zhou, Lei Liang, Dan He, and Zhili Sun. A sevice-oriented broker for bulk data transfer in cloud computing. In *Grid and Cooperative Computing (GCC), 2010 9th International Conference on*, pages 264 –269, nov. 2010. 20, 27

[54] Owen Rogers and Dave Cliff. A financial brokerage model for cloud computing. *Journal of Cloud Computing: Advances, Systems and Applications*, 1(1):2, 2012. 20, 27

[55] Hao Li, Jianhui Liu, and Guo Tang. A pricing algorithm for cloud computing resources. In *Proceedings of the 2011 International Conference on Network Computing and Information Security - Volume 01*, NCIS '11, pages 69–73, Washington, DC, USA, 2011. IEEE Computer Society. 20, 27

[56] Elizabeth Chang, Tharam Dillon, and Farookh K. Hussain. *Trust and Security in Service-Oriented Environments*. John Wiley & Sons, Ltd, 2006. 31

[57] Xiaoyong Tang, Kenli Li, Zeng Zeng, and Bharadwaj Veeravalli. A novel security-driven scheduling algorithm for precedence constrained tasks in heterogeneous distributed systems. *IEEE Transactions on computers*, 60(17):1017–1029, 2011. 31

[58] John Alcock Ian Mitchell. Cloud security: The definitive guide to managing risk in the new ict landscape. White paper, Fugistu Services Ltd, 2011. 34

[59] R Preston McAfee and John McMillan. Auctions and bidding. *Journal of Economic Literature*, 25(2):699–738, June 1987. 35

[60] V. Bhaskar and Ichiro Obara. Belief-based equilibria in the repeated prisoners' dilemma with private monitoring. *Journal of Economic Theory*, 102(1):40 – 69, 2002. 39, 40

[61] R. Jain, D. Chiu, and W. Hawe. A quantitative measure of fairness and discrimination for resource allocation in shared computer systems, 1998. http://www.citebase.org/abstract?id=oai:arXiv.org:cs/9809099. 42

[62] A. Mas-Colell, M.D. Whinston, and J.R. Green. *Microeconomic Theory*. Oxford University Press, 1995. 53, 57, 59

[63] William Sears, Zhen Yu, and Yong Guan. An adaptive reputation-based trust framework for peer-to-peer applications. In *Proceedings of the Fourth IEEE International Symposium on Network Computing and Applications*, NCA '05, pages 13–20, Washington, DC, USA, 2005. IEEE Computer Society. 56, 57

[64] C. Aliprantis and S. Chakrabarti. *Games and Decision Making*. Oxford University Press, 2010. 57, 58

[65] Wenjing Wang, M. Chatterjee, and K. Kwiat. Attacker detection game in wireless networks with channel uncertainty. In *Communications (ICC), 2010 IEEE International Conference on*, pages 1 –5, may 2010. 58

[66] S. Sengupta, M. Chatterjee, and K.A. Kwiat. A game theoretic framework for power control in wireless sensor networks. *Computers, IEEE Transactions on*, 59(2):231 –242, feb. 2010. 58

[67] Kai Shen, Shoubao Yang, Wei Chen, Xiaoqian Liu, and Bin Wu. Balancing risk and price: An opportunity-cost approach for job scheduling in the grid

market. In *Proceedings of the Sixth International Conference on Grid and Cooperative Computing*, GCC '07, pages 521–527, Washington, DC, USA, 2007. IEEE Computer Society. 58, 59

[68] Abhinay Muthoo. *Bargaining theory with applications*. Cambridge University Press, New York, NY, USA, 1999. 85, 88, 89, 90, 91, 92

[69] Zhangyu Guan, Dongfeng Yuan, and Haixia Zhang. Novel coopetition paradigm based on bargaining theory or collaborative multimedia resource management. In *PIMRC*, pages 1–5. IEEE, 2008. 85

[70] Xiaodong Yan, Zhijuan Wang, Weijun Cheng, and Liping Zhu. A pricing strategy model based on economy theory in mobile grids. In *Wireless Communications, Networking and Mobile Computing, 4th International Conference on*, WiCOM '08, pages 1–4, 2008. 85

[71] Antonella Di Stefano and Corrado Santoro. An economic model for resource management in a grid-based content distribution network. *Future Gener. Comput. Syst.*, 24(3):202–212, March 2008. 85

[72] Preetam Ghosh, Kalyan Basu, and Sajal K. Das. A game theory-based pricing strategy to support single/multiclass job allocation schemes for bandwidth-constrained distributed computing systems. *IEEE Trans. Parallel Distrib. Syst.*, 18(3):289–306, March 2007. 85

[73] Riky Subrata, Albert Y. Zomaya, and Bjorn Landfeldt. A cooperative game framework for qos guided job allocation schemes in grids. *IEEE Trans. Comput.*, 57(10):1413–1422, October 2008. 86

[74] B.J.S. Chee and J. Curtis Franklin. *Cloud Computing: Technologies and Strategies of the Ubiquitous Data Center*. An Auerbach book. Taylor & Francis, 2009. 86

[75] Xiren Cao. Preference functions and bargaining solutions. In *Decision and Control, 1982 21st IEEE Conference on*, volume 21, pages 164–171, 1982. 91, 94

[76] Francois Berenger, Camille Coti, and Kam Y. J. Zhang. Par: a parallel and distributed job crusher. *Bioinformatics*, 26(22):2918–2919, 2010. 107

[77] E. N. Cáceres, H. Mongelli, L. Loureiro, C. Nishibe, and S. W. Song. Performance results of running parallel applications on the integrade. *Concurr. Comput. : Pract. Exper.*, 22(3):375–393, March 2010. 107

[78] Santos E. L. Neto, L. E. F. Tenrio, E. J. S. Fonseca, S. B. Cavalcanti, and J. M. Hickmann. Parallel visualization of the optical pulse through a doped optical fiber. In *In Proceedings of Annual Meeting of the Division of Computational Physics (abstract)*, 2001. 107

[79] James Cowie, Bruce Dodson, R. Marije Elkenbracht-Huizing, Arjen K. Lenstra, Peter L. Montgomery, and Jörg Zayer. A world wide number field sieve factoring record: On to 512 bits. In *Proceedings of the International Conference on the Theory and Applications of Cryptology and Information Security: Advances in Cryptology*, ASIACRYPT '96, pages 382–394, London, UK, UK, 1996. Springer-Verlag. 108

[80] Doruk Bozdag, Catalin C. Barbacioru, and Umit V. Catalyurek. Parallel short sequence mapping for high throughput genome sequencing. In *Pro-*

ceedings of the 2009 IEEE International Symposium on Parallel&Distributed Processing, IPDPS '09, pages 1–10, Washington, DC, USA, 2009. IEEE Computer Society. 108

[81] Dimitri Bertsekas and Robert Gallager. *Data networks*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1987. 109, 123, 125

[82] Bharadwaj Veeravalli and Nukala Viswanadham. Suboptimal solutions using integer approximation techniques for scheduling divisible loads on distributed bus networks. *IEEE Transactions on Systems, Man, and Cybernetics, Part A*, 30(6):680–691, 2000. 111

[83] Albert Greenberg, James R. Hamilton, Navendu Jain, Srikanth Kandula, Changhoon Kim, Parantap Lahiri, David A. Maltz, Parveen Patel, and Sudipta Sengupta. Vl2: A scalable and flexible data center network. In *Proc. of the ACM SIGCOMM 2009 conference on Data communication (SIGCOMM'09)*, Barcelona, Spain, August 2009. 115

[84] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. A scalable, commodity data center network architecture. In *Proc. of the ACM SIGCOMM 2008 conference on Data communication (SIGCOMM'08)*, pages 63–74, Seattle, Washington, USA, August 2008. 115

[85] P. Sugavanam, H. J. Siegel, A. A. Maciejewski, M. Oltikar, A. M. Mehta, R. Pichel, A. Horiuchi, V. Shestak, M. Al-Otaibi, Y. G. Krishnamurthy, S. A. Ali, J. Zhang, M. Aydin, K. Guru P. Lee, M. Raskey, and A. J. Pippin. Robust static allocation of resources for independent tasks under makespan

and dollar cost constraints. *Journal of Parallel and Distributed Computing*, 67(4):400–416, 2007. 118

[86] S. T. McCormick and M. L. Pinedo. Scheduling n independent jobs on m uniform machines with both flowtime and makespan objectives: A parametric analysis. *ORSA Journal on Computing*, 7(1):63–77, 1995. 118

[87] Alexandru Iosup, Ozan Sonmez, Shanny Anoep, and Dick Epema. The performance of bags-of-tasks in large-scale distributed systems. In *Proceedings of the 17th international symposium on High performance distributed computing*, HPDC '08, pages 97–108, New York, NY, USA, 2008. ACM. 131, 132

[88] Cosimo Anglano, Massimo Canonico, Marco Guazzone, Marco Botta, Sergio Rabellino, Simone Arena, and Guglielmo Girardi. Peer-to-peer desktop grids in the real world: The sharegrid project. In *Proceedings of the 2008 Eighth IEEE International Symposium on Cluster Computing and the Grid*, CCGRID '08, pages 609–614, Washington, DC, USA, 2008. IEEE Computer Society. 132

[89] D. Turgay Altilar and Yakup Paker. An optimal scheduling algorithm for parallel video processing. In *In IEEE Int. Conference on Multimedia Computing and Systems. IEEE Computer*. Society Press, 1998. 140

[90] Ganesh Neelakanta Iyer, Bharadwaj Veeravalli, and Sakthi Ganesh Krishnamoorthy. On handling large-scale polynomial multiplications in compute cloud environments using divisible load paradigm. *Aerospace and Electronic Systems, IEEE Transactions on*, 48(1):820 –831, jan. 2012. 140, 148

[91] Y. Ji, D.C. Marinescu, W. Zhang, X. Zhang, X. Yan, and T.S Baker. A model-based parallel origin and orientation refinement algorithm for cryotem and its application to the study of virus structures. *Journal of Structural Biology*, 154(1):1–19, 2006. 140

[92] Arnaud Legrand, Alan Su, and Frédéric Vivien. Minimizing the stretch when scheduling flows of biological requests. In *Proceedings of the eighteenth annual ACM symposium on Parallelism in algorithms and architectures*, SPAA '06, pages 103–112, New York, NY, USA, 2006. ACM. 140

[93] Siani Pearson and Azzedine Benameur. Privacy, security and trust issues arising from cloud computing. In *Proceedings of the 2010 IEEE Second International Conference on Cloud Computing Technology and Science*, CLOUDCOM '10, pages 693–702, Washington, DC, USA, 2010. IEEE Computer Society. 140

[94] Veeravalli Bharadwaj, Thomas G. Robertazzi, and Debasish Ghose. *Scheduling Divisible Loads in Parallel and Distributed Systems*. IEEE Computer Society Press, Los Alamitos, CA, USA, 1996. 148, 150

[95] Bharadwaj Veeravalli, Xiaolin Li, and Chi Chung Ko. On the influence of start-up costs in scheduling divisible loads on bus networks. *IEEE Trans. Parallel Distrib. Syst.*, 11(12):1288–1305, December 2000. 151

Author's Publications

[**Book Chapter**] **Ganesh Neelakanta Iyer** and Bharadwaj Veeravalli, "Design and Analysis of Broker-Mediated Cloud Aggregation Mechanisms Using Markovian Queues for Scheduling Bag-of-Tasks" To Appear as a Book Chapter in *Large Scale Network-centric Computing Systems*, A. Y. Zomaya and H. Sarbazi-Azad, Eds., John Wiley & Sons, Hoboken, NJ, USA 2012.

[**Journal**] **Ganesh Neelakanta Iyer**, Bharadwaj Veeravalli and Sakthi Ganesh Krishnamoorthy, "On Handling Large Scale Polynomial Multiplications in Compute Cloud Environments using Divisible Load Paradigm.", IEEE Transactions on Aerospace and Electronic Systems, vol.48, no.1, pp.820-831, January 2012.

[**Conference**] **Ganesh Neelakanta Iyer**, Ramkumar Chandrasekaran and Bharadwaj Veeravalli, "Auction-based vs. Incentive-based Multiple-Cloud Orchestration Mechanisms", IEEE International Conference on Communication, Networks and Satellite (COMNETSAT 2012), July 2012 (Accepted).

[**Conference**] **Ganesh Neelakanta Iyer** and Bharadwaj Veeravalli, "On the Resource Allocation and Pricing Strategies in Compute Clouds Using Bargaining Approaches", IEEE International Conference on Networks (ICON 2011), Singapore, December 2011.

[**Journal**] Lingfang Zeng, **Ganesh Neelakanta Iyer** and Bharadwaj Veeravalli, "Priority Based Task Scheduling for Bag-of-Tasks Applications in Cloud Computing Environments", Journal of Parallel and Distributed Computing (JPDC), Elsevier 2012 (Pending Decision Status).

[**Journal**] **Ganesh Neelakanta Iyer**, Bharadwaj Veeravalli and Ramkumar Chandrasekaran, "Broker based Cloud Service Arbitrage Mechanisms using Sealed-bid Double Auctions and Incentives", IEEE Transactions on Network and Service Management, IEEE 2012 (Under Review).

[**Journal**] **Ganesh Neelakanta Iyer**, Li Xiao and Bharadwaj Veeravalli, "Risk Aware Cloud Broker Arbitrage Mechanism Based on Trust and Dynamic Pricing Strategies", IEEE Transactions on Parallel and Distributed Systems, IEEE 2012 (Under Review).

[**Journal**] **Ganesh Neelakanta Iyer** and Bharadwaj Veeravalli, "Taxonomy of Broker Mediated Cloud Services Architectures", IEEE Transactions on Computers, IEEE 2012 (Under Review).