Study on Combinatorial Auction Mechanism for Resource Allocation in Cloud Computing Environment

Ikki Fujiwara

DOCTOR OF PHILOSOPHY

Department of Informatics
School of Multidisciplinary Sciences
The Graduate University for Advanced Studies

2011

A dissertation submitted to
the Department of Informatics,
School of Multidisciplinary Sciences,
The Graduate University for Advanced Studies (SOKENDAI)
In partial fulfillment of the requirements for the degree of Doctor of Philosophy

Advisory Committee

Prof. Kento Aida	National Institute of Informatics, SOKENDAI
Assoc. Prof. Isao Ono	Tokyo Institute of Technology
Prof. Kenichi Miura	National Institute of Informatics, SOKENDAI
Assoc. Prof. Michihiro Koibuchi	National Institute of Informatics, SOKENDAI
Prof. Shigeo Urushidani	National Institute of Informatics, SOKENDAI

Published in March 2012

Acknowledgements

My time as a graduate student at SOKENDAI and NII has been supported by many kind people. This page attempts to recognize those who have been most helpful along the way.

First of all, I am deeply indebted to my supervisor, Professor Dr. Kento Aida. His advice, support and kind encouragement on both professional and private issues have effectively guided me through this winding way. Furthermore, he always led my papers to clear, smart and persuasive ones. Without him this thesis would not have been possible.

At the same time, I am also indebted to my advisor, Professor Dr. Isao Ono. His experience, technique and insight into both economics and engineering have immensely supported me in designing the mechanism and developing the simulator.

Thanks to all the members of the Aida Laboratory. Dr. Hao Sun has regularly shared a broad range of helpful technical information with me. I specifically appreciate his cooperation in debugging the most difficult problem during development of the simulator. Dr. Md. Shamim Akhter directed me to the world of researcher based on his extensive experience over the world. Dr. Kiyoshi Osawa, Kazushige Saga, Keigo Sakamoto, Yuya Hashimoto, Jared Ravetch, Kosuke Naruse and other members of the Aida Laboratory shared their precious time with me and gave me many constructive comments on my work.

Special thanks go to my Ph.D. examiners, Professor Dr. Kenichi Miura, Professor Dr. Michihiro Koibuchi, and Professor. Dr. Shigeo Urushidani. Each devoted significant time and effort to my thesis, and their suggestions and comments led to substantial improvement in the final product.

Outside the laboratory, I have benefited from time spent with friends at SOKENDAI. Interdisciplinary discussions with these friends have opened my eyes greatly than expected before. Finally, I thank my family for all aspects of my life.

Abstract

This thesis proposes a combinatorial auction-based marketplace mechanism for cloud computing services, which allows users to reserve arbitrary combination of services at requested timeslots, prices and quality of service. The proposed mechanism helps enterprise users build workflow applications in a cloud computing environment, specifically on the platform-as-a-service, where the users need to compose multiple types of services at different timeslots.

The proposed marketplace mechanism consists of a forward market for an advance reservation and a spot market for an immediate allocation of services. Each market employs mixed integer programming to enforce a Pareto optimum allocation with maximized social economic welfare, as well as double-sided auction design to encourage both users and providers to compete for buying and selling the services.

A marketplace simulator, named W-Mart, is specially developed for this thesis. It implements the proposed mechanism on Java platform being powered by CPLEX, the state-of-the-art MIP solver. W-Mart is designed after the multi-agent virtual market system U-Mart, and is also capable to deal with human agents and machine agents at the same time.

Three experiments are carried out by means of multi-agent simulations. First, the accuracy of the combinatorial allocation scheme is validated. The result

demonstrates that it works properly. Second, the overhead of the proposed market mechanism including MIP solver is assessed. The result shows that the overhead is acceptable to deal with an expected number of participants within the proposed trading schedule. Third, the performances of four types of market mechanisms are extensively evaluated. The results clarify that (1) the proposed forward/combinatorial mechanism outperforms other non-combinatorial and/or non-reservation (spot) mechanisms in both user-centric rationality and global efficiency, (2) running both a forward market and a spot market improves resource utilization without disturbing advance reservations, and (3) the users' preference between the forward market and the spot market affects the performance of whole marketplace significantly in tight demand/supply conditions.

Contents

List of Figures

List of Tables

(This page is intentionally left blank)

Chapter 1 Introduction

1.1 Motivation

Throughout its 60 years of history, computer systems have been evolved in a spiral way of integration and distribution. They experienced a transition from centralized, massive, shared mainframes in 1970s to decentralized, handy, private PCs in 1990s. In 2010s, nevertheless, they are again moving into consolidated, shared machines invisible to users: so-called cloud computing systems.

The basic idea of cloud computing is to deliver computational resources as services across the internet. The users don't need to invest in a huge computer system to do their business; instead, they can purchase the cloud computing services on demand. The underlying hardware is generally hosted in huge datacenters armed with sophisticated virtualization techniques to realize high-level scalability, agility and availability. "This elasticity of resources, without paying a premium for large scale, is unprecedented in the history of IT", Armbrust said [1].

Consequently, building enterprise systems on cloud computing platform is becoming increasingly popular in these days. In contrast to a conventional on-premise computing system, to which the user has to invest in dedicated hardware and software, the cloud computing system delivers virtualized hardware and software resources to users on demand via the internet, generally in a pay-per-use manner. It significantly reduces the cost for deploying and maintenance

of enterprise systems.

The cloud is described as a three-tier structure, namely Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS) from low-layer to high-layer. Recently PaaS is evolving rapidly as a software development/deployment environment for enterprise systems. For example, Microsoft Windows Azure [2] provides .NET development environment and SQL service, whereas Google App Engine [3] provides Python and Java development environment with key-value store service. A developer chooses an appropriate service among available ones to build his customized system.

An enterprise system is generally consists of multiple subsystems to model a complex real business. The subsystems are often provided by service on the internet, or PaaS, and the developer of the enterprise system needs to choose appropriate PaaS providers to develop an efficient system. As the number of PaaS provider will increase, a challenging issue is how to choose an appropriate combination of services, or PaaSes, to build a complex enterprise system. Furthermore, the enterprise system has strict requirements for the quality of service (QoS) as well as a budget limitation. Unfortunately no practical workaround to this complex problem is realized so far in the cloud computing environment.

This kind of problem has long been discussed as a resource allocation problem on a distributed computing system [4]. The problem is typically described as a sort of optimization problem and tends to have computational complexity to obtain optimum solutions. A market-based approach is a promising methodology to deal with the complexity while satisfying budget limitation [5,6]. Specifically a combinatorial auction-based approach is recently evolved because of its ability to optimize allocation of a bundle of multiple goods [7,8]. However, the previous work does not satisfy requirements of resource allocation in the enterprise system composed of multiple cloud services, as discussed in Section 2.3.

1.2 Research Objectives

This thesis proposes a marketplace mechanism for resource allocation to develop an enterprise system in cloud computing environment. The proposed mechanism performs resource allocation between users and PaaS providers using combinatorial double-sided auction. In other words, both users and PaaS providers place buying/selling orders of resources, and the market mechanism decides resource allocations by means of auctions. The user's enterprise system often consists of multiple resources, thus, the user is able to order arbitrary combination of services for future use at desired time, price and QoS requirements. The proposed mechanism employs mixed integer programming (MIP) to obtain optimal resource allocation enforcing a Pareto optimum outcome as well as a fair trading.

1.3 Scope and Limitations

Despite the emphasis on "enterprise PaaS" stories told above, the proposed market mechanism can also be adopted to the other cloud services as IaaS or SaaS, and even to a broader variation of combinatorial resource allocation/scheduling problems. The mechanism itself is a versatile, implementation-independent design of a combinatorial marketplace; therefore its application is not limited to computing services.

The scope of this thesis is, however, limited to a fundamental design of market mechanism and its evaluation. Physical considerations such as network delay and provisioning time are intentionally ignored; they may be treated as separate services in practice. Security requirements are also assumed to be satisfied, as well as governance and compliance issues. Currency used in the marketplace should be any real currency as EUR/USD/JPY/etc. Use of virtual currency involves another research topic of ecosystem management, and it is out of the scope of this thesis.

1.4 Contributions

Cloud computing is a promising paradigm of computing in the next decade. As the number of stakeholders is increasing rapidly, the role of marketplace should widely be accepted. Providing a fundamental design of cloud-friendly market mechanism, this thesis contributes a blueprint for pioneers to establish a cloud service marketplace. The marketplace will bring three benefits to the stakeholders: (1) it will help users discover and acquire appropriate services at a reasonable cost amongst many providers; (2) it will help providers meet potential users and make maximum utilization of their own resources; and additionally (3) it will give the operator of marketplace supremacy over the computing industry. The author hopes this research will promote substantial competitions among users, providers and operators of cloud marketplaces in the future.

1.5 Outline of the Thesis

The remainder of this thesis is organized as follows. Chapter 2 provides a common understanding of market institutions and auction theory, as well as preceding instances of market-based management techniques for computing resource management. Additionally, the applications of market mechanism in other disciplines are also mentioned. Chapter 3 introduces the basic concepts of cloud computing and its use cases for enterprise systems. Four requirements are clarified to design an efficient marketplace from both practical and theoretical points of view. Chapter 4 presents the design and algorithms of the cloud service marketplace. First it outlines the institution of the marketplace and its participants. Next it formulates the service allocation scheme as a mixed integer program (MIP), as well as the pricing scheme which extends the concept of K-pricing. Chapter 5 illustrates the design and implementation of simulator named W-Mart. It translates the conceptual

model into a software system as hybrid architecture of a multi-agent simulation framework and a general-purpose optimization framework. Chapter 6 describes and analyzes the results of three experiments which are carried out to evaluate the proposed mechanism. Section 6.1 validates the combinatorial allocation and confirms that the implemented mechanism properly works as expected. Section 6.2 evaluates the overhead of the proposed mechanism and shows that it is acceptable in the cloud service market. Section 6.3 evaluates the performance of the proposed mechanism and concluded that it outperforms other types of market mechanisms. Chapter 7 concludes with a summary, an overview of open questions and an outlook on future world with the cloud computing marketplace.

(This page is intentionally left blank)

Chapter 2 Background and Related Work

This research stands on an interdisciplinary area of computer science and economics. This Chapter first studies thoroughly on market mechanism, mainly focusing on auction design. Next, it reviews the preceding researches on market-based management of computer systems. Finally, it surveys other applications of market mechanism in the real world, discussing the possibility of applying their market models to the cloud computing systems.

2.1 Market Mechanisms

Markets are institutions through which exchange of resources is facilitated. A French economist Antoine Augustin Cournot (1801-1877) so defined the market that "economists understand by the term Market, not any particular market place in which things are bought and sold, but the whole of any region in which buyers and sellers are in such free intercourse with one another that the prices of the same goods tend to equality easily and quickly" [9]. Nowadays the notion of market is not limited to a geological region but also includes a virtual structure that allows participants to exchange any type of goods, services and information [10]. The most essential role of a market, in my opinion, is to aggregate information on supply and demand; hence it is quite natural to use market mechanism for promotion,

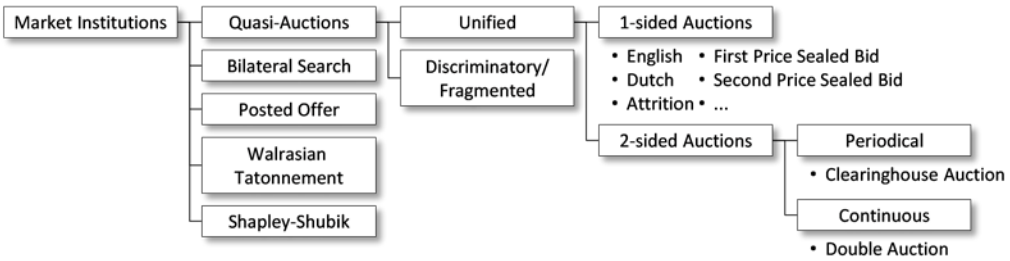


Figure 2.1 A taxonomy of market mechanisms

procurement, allocation and/or scheduling of computing resources.

There exists a broad range of market mechanisms invented so far and also exists a variety of taxonomy of them. For instance, Figure 2.1 is Freidman's taxonomy of market institutions [11]. Well, which particular kind of market mechanism among them is the best for trading computing resources? To answer it, next two chapters study a broad outline of the market mechanisms and, in particular, auction mechanisms.

2.1.1 A Taxonomy of Market Mechanisms

First, let us classify some of the typical market mechanisms following the line of Buyya's taxonomy [12].

A. Commodity Market

What we first imagine with the word "market" may be a commodity market, where the sellers set the price for merchandise and the buyers pay money to get it. An example is a farmer in a marketplace selling tomatoes at $3 for 1kg; you may purchase 2kg of them and pay $6 – that is the commodity market. The price is pre-determined by the seller and does not change (flat-rate) or changes over time (variant-rate) based on the balance of supply and demand. The payment is usually proportional to the quantity of the merchandise. The commodity market model is

quite straightforward and widespread; thus it can be the baseline of the computing resource marketplace.

B. *Posted Price*

Within the commodity market the seller may have a "special offer" like discounted merchandise or free gifts to attract customers. The posted offer model allows suppliers to advertise openly such an irregular offerings with specified conditions. This is an extension of the commodity market model and is also very common in our daily life.

C. *Bargaining*

In some countries the price of merchandise is not shown at all; instead the customer and the shopkeeper negotiate for a mutually agreeable price. Usually the seller starts from a higher price and the buyer starts from a lower price. The final payment is fixed when they agree; otherwise the trading fails. The bargaining model is troublesome but practical when it is difficult for the seller to determine the price based on the supply and demand.

D. *Tendering*

Imagine a public procurement by government office, for example, construction of a road. The government first announces the detail of the required product. Construction companies then examine it and decide whether to make a bid for it or to ignore it. The government collects the bids, chooses the most suitable (usually the cheapest) one, sends a tender to the selected constructor, and pays money when the product is completed. That is the tendering model and is mainly used to acquire a customized product or service at an appropriate price.

E. *Proportional Sharing*

What can we do if demand exceeds supply? One solution is to raise the price and the other is to reduce the quantity of allocation. Proportional sharing model does the latter; it distributes the resource in proportion to the payments given by the buyers. For instance, the seller has 50kg of rice and the buyer A pays $60 and the buyer B pays $40; then the seller gives buyer A 30kg and buyer B 20kg of rice. This can be seen as a modification of commodity market, where seller gives up to control the price and buyers cooperate to share the resource in a fair manner.

F. *Auction*

What if there are many buyers for single merchandise which cannot be divided? The only solution is to adjust the price so that an only one buyer wishes to pay that price. The auction model is often used in such situations to adjust the price by competing biddings among the buyers. The tendering model can be seen as another kind of auctions where the competition occurs among the sellers. There are many other kinds of auction mechanisms described in the next section. The auction model is the most sophisticated way to determine the price in terms of fairness and

Table 2-1 Characteristics of typical market mechanisms

	#Seller : #Buyer	**Price leader**
Commodity market	1 : N	Seller
Posted price	1 : N	Seller
Bargaining	1 : 1	Both
Tendering	N : 1	Buyer
Proportional share	1 : N	Buyer
Auction	Vary	Auctioneer

optimality; thus it is the most promising way to design a computing resources marketplace.

2.2 Auction Theory

Auction is a type of market mechanism where the price is neither preset nor arrived at by negotiation, but is discovered through the process of competitive bidding. More formally speaking, an auction is a "protocol that allows agents to indicate their interest in one or more resources and that uses these indications of interest to determine both an allocation of resources and a set of payments by the agents" [13]. Auction allows multiple buyers to negotiate with a single seller by submitting bids through an auctioneer, or vice versa. The auctioneer sets the rules of the auction and acts as an arbitrator. Negotiation continues until a single clearing price is reached. Thus, auction regulates supply and demand based on the competition of bidding price among sellers and/or buyers.

Auction is a powerful tool to resolve a competitive situation of resource acquirement. There is virtually infinite variety of auction mechanisms in the world. This thesis classifies them from two points of view: the number of goods and the

Table 2-2 Taxonomy of auction design

		#Sellers : #Buyers	
		1 : N or M : 1	**M : N**
#Goods	**One**	Single-good single-sided (Section 2.2.1)	Single-good double-sided (Section 2.2.2)
	Many	Combinatorial (Section 2.2.3)	

number of participants (i.e. sellers and buyers). The former means that how many kinds of good (or merchandise, service, etc.) are traded in an auction at the same time. The latter implies that on which side of participants the competition occurs. Table 2-2 summarizes the classification and the sections below review typical types of auction from above two points of view.

2.2.1 Single-good Single-sided Auctions

A. English auction

Perhaps the reader imagine the English auction simply by the word "auction", because the English auction is widely used in both traditional auction houses and today's internet auctions. In English auctions, there is one seller and the competition occurs among multiple buyers. The procedure is (1) the auctioneer first declares the initial price of the good, (2) the buyers then monotonically raises the price by making their bids, and (3) the auctioneer finally stops the biddings when no more biddings are made or when a fixed time is reached. The final bidder with the highest price becomes the winner and he must purchase the good at that price.

B. Japanese auction

In Japanese auctions, not the buyer but the auctioneer raises the price. The procedure is (1) the auctioneer sets the initial price and starts raising it periodically, (2) in each period a buyer must declare whether he remains in or drops out of the auction, and (3) the auction finishes when only one buyer remains in. The final remained buyer becomes the winner and he must purchase the good at the final price. A dropped buyer cannot come back to the same auction.

C. Dutch auction

In Dutch auctions, the price goes down from high to low. The procedure is (1) the auctioneer declares a high initial price and starts a wall clock that indicates the

descending price, (2) a buyer pushes a button at an arbitrary time to stop the wall clock, and (3) the auction finishes immediately. The first buyer pushing the button becomes the winner and he must buy purchase the good at that price. The Dutch auction is used in the Amsterdam flower market as it is particularly suitable to determine the winner as quick as possible.

D. *Sealed-bid auctions*

The biddings can be secret from other buyers. In sealed-bid auctions, unlike the three types of auctions discussed above, the biddings are not open to the public. The procedure is (1) the buyers submit their "sealed" biddings directly to the auctioneer, (2) the auctioneer then compares these biddings, and (3) the buyer with the highest bid becomes the winner and he must purchase the good. The price at which the winner purchase the good is determined separately in several ways: (a) in first-price auctions the winner's due is equal to his own bid, (b) in second-price auctions the winner's due is equal to the second-highest bid (i.e. the highest rejected bid), and (c) in kth-price auctions the winner's due is equal to the kth-highest bid.

The second-price auctions are also used in combination with the English auctions. For instance, in the internet auctions like Yahoo! Auctions, (1) the buyer tells his reservation price to a proxy agent, (2) the proxy agent then keeps bidding above the previous highest bid until the reservation price is reached, and (3) the auction finishes at a pre-defined time. The highest bidder becomes the winner and he pays the price just one unit above the second-highest bid. This type of auction can be seen as a combination of the English auction and the second-price sealed-bid auction.

E. *Reverse Auctions*

The above four sections have considered situations with one seller and multiple buyers. We can consider the opposite: auctions with multiple sellers and one buyer.

Such a family of auctions is called reverse auctions. For instance, in a public procurement, there is one buyer (e.g. a local government office) and multiple sellers (e.g. construction companies) competing each other to provide the lowest price for a good (e.g. building). The discussion on the non-reverse auctions can also be applied to the reverse auctions by negating the prices (a negative payment means an earning) and swapping the word "seller" for "buyer", "highest" for "lowest" and so on, without loss of generality. Therefore the sections below will not discuss reverse auctions any further.

2.2.2 *Single-good Double-sided Auctions*

There may be many buyers and many sellers at the same time for a single kind of merchandise. This setting is called double-sided auctions or exchanges. A typical example is the stock market, in which many people are participating to buy or sell a company's stock. Note that the double-sided auction causes two kinds of competition at the same time: one is on the buyer's side (a buyer competes against other buyers) and the other is on the seller's side (a seller competes against other sellers). It is different from the internet auctions like Yahoo! Auctions where the sellers do not directly compete with each other.

There are primarily two approaches to implement the single-good double-sided auctions: the continuous double auction (CDA) and the periodic double auction (PDA, also known as the clearinghouse auctions). In both the CDA and the PDA, a participant places orders at any time and as many times as he want. Each order includes a price (maximum for buying or minimum for selling) and a quantity (positive for buying or negative for selling). The auctioneer then registers the incoming orders on an order book.

The difference between the CDA and the PDA is the time when the trading occurs. In the CDA, as soon as the order comes, the auctioneer attempts to match it

Before			After		
#Sell	Price	#Buy	#Sell	Price	#Buy
4	$9	6	4	$9	
0	$8	0	0	$8	
0	$8	0	0	$8	
2	$6	0	2	$6	
0	$5	4	0	$5	
6	$4	6		$4	2
0	$3	3		$3	3
3	$2	5		$2	5
5	$1	2		$1	2

Figure 2.2 Example order book of periodical double-sided auction

against the orders put on the order book. For example, an incoming buy order for 10 units may be matched against an existing sell order for 4 units and another sell order for 6 units, as long as the buying price exceeds the selling price. In cases of partial matches, the remaining units (either of the incoming order or of the existing orders) are put back on the order book. For example, when a sell order for 13 units is coming and two buy orders – one for 4 units and another for 6 units – are existing on the order, 10 units of the sell order is matched against the buy orders and the remaining 3 units are put on the order book as a sell order.

In the PDA, in contrast, the auctioneer does not attempt to match the order at the time it comes; instead he simply put it on the order book. Then, at some pre-defined clock time, he attempts to match the orders as much as possible. The

Itayose[1] algorithm is often used for matchmaking, by which he ranks the orders according to their price and determines the point at which supply meets demand. Figure 2.2 depicts a snapshot of an order book before and after the trading occurs. In this example 14 units are traded and the remaining ones are left on the order book.

2.2.3 Combinatorial Auctions

Combinatorial auction is the most generic form of auctions, which allows different kinds of good to be traded at once, unlike the single-good auctions discussed in the above sections. This is particularly meaningful for the agents (buyers and/or sellers) whose valuations depend on the set of goods they deal with, rather than the good itself. For example, to make use of a computer, you need to procure not only the computer itself but also its operating system and perhaps an internet connection. Real-world applications of the combinatorial auctions so far include the auctions for radio spectrum, energy, shipping paths, and corporate procurement [13].

More formally, consider a set of agents $N = \{1, \cdots, n\}$ and a set of goods $G = \{1, \cdots, m\}$. Let $v = (v_1, \cdots, v_n)$ denote the valuation functions of the different agents, where valuation for each $i \in N, v_i : 2^G \to \mathbb{R}$. There is an assumption that there are no externalities. Specifically, it is asserted that an agent's valuation depends only on the set of goods he wins; that is, we do not consider such an agent who also cares about the allocations and payments of the other agents.

The agents in the combinatorial auctions have nonadditive valuation functions. There are two kinds of nonadditivity: substitutability and complementarity.

The substitutability means that the combined value of multiple goods is less

[1] (Japanese) Literally "gather the board".

than the sum of their individual values. For example, imagine you want to buy a car and your option is a Toyota and a Renault. Since you cannot drive both of them at the same time, your satisfaction of having both is less than twice as much as having one of them; in other words, these two cars are partial substitutes for each other. If two goods are strict substitutes, their combined value is equals to the value for either one – in this case these items can also be seen as multiple units of a single kind of good.

The complementarity means that the combined value of multiple goods is greater than the sum of their individual values. For example, imagine you are commanded to travel from Tokyo to Pyongyang via Beijing; you need the tickets for both flights – not only one of them – otherwise you cannot accomplish your mission. A business process is much like this; the value of corporation is created not by each job alone but by the combined outcome of whole company.

The advantage of the combinatorial auctions is their ability to simplify the agent's strategy. If there is no combinatorial auction, for example in the complemental case, an agent needs to arrange a bundle of multiple goods in a one-by-one manner at the risk of incompletion (so-called "exposure risk"). This is a risky and complex task for the agent. With combinatorial auctions, in contrast, an agent needs to simply express his requirement for a bundle to the auctioneer. There is no risk of incompletion since the allocation is guaranteed to be all-or-nothing. Essentially the combinatorial auction eliminates the complexity from the agent and instead transfers it to the auctioneer. "Indeed, these auctions have been the subject of considerable recent study in both economics and computer science" [13].

2.3 Market-based Computing Resource Management

2.3.1 In Literature

The idea of using markets and pricing computer resources is quite old. Pricing policies received considerable attention at the dawn of multi-user time sharing systems. Papers in the late 1960's were dedicated to automatic pricing policies for computer time [14,15]. For modern researches in this area, Buyya et al. provides some comprehensive surveys [12,16,17] as well as grid/cloud computing toolkits and simulators [18,19].

Spawn [20] by Waldspurger et al. is the first implementation of an auction-based management system for distributed computing resources. It aims at utilizing idle CPU times in a network of workstations. Spawn employs periodical, single-good, single-sided, sealed-bid, second-price auctions. The buyers are end-users, the seller is the owner of the workstation, the auctioneer is the Spawn system running on the seller's workstation, and the good is its CPU time. The trading procedure is (1) a buyer finds a seller and bids for the CPU time of the workstation, (2) the seller determines the winner, and (3) the winning buyer runs his program on the seller's workstation. Note that the auctions are distributed; i.e. there are as many auctions as there are sellers in the network. The buyer needs to find an appropriate seller using nearest-neighbor connections provided. The Spawn's scheme does not fit the enterprise cloud environment because its distributed nature of multiple auctioneers (1) prevents the system from scaling beyond a local network, and (2) prevents the buyers from bidding on a combination of multiple resources from different sellers.

POPCORN [21,22] by Nisan et al. basically extends the scope of Spawn's "network of workstations" idea from local network to global network. It aims to be "an infrastructure for globally distributed computation over the whole Internet" [21].

POPCORN employs a periodical, single-good, single- or double-sided, sealed-bid, second-price auction. The buyers are Java programs written using POPCORN framework, the sellers are people using Java-enabled browsers, the auctioneer is an independent market service on the Internet, and the good is a right to run a Java applet ("Computelet"). The trading procedure is (1) a programmer writes a program using POPCORN framework and runs it on a local computer as a buyer, (2) the buyer connects to the market and bids for execution of Computelets, (3) a seller visits a web page where a POPCORN applet is embedded, (4) the applet connects to the market and asks for execution of a Computelet, (5) the market determines the winners, and (6) the winning buyer spawns Computelets and sends them to the winning sellers so that they execute the Computelets on their browsers and return the results back to the local computer. The POPCORN's scheme overcomes the scalability problem of Spawn's; however, it still does not fit the enterprise cloud environment because it does not support multiple kinds of resources and, of course, combination of them.

G-commerce [23,24] by Wolski et al. compares two different market models – commodities market and auctions – for resource allocation in Grid computing environment. It on the one hand employs a commodities market with price adjustment scheme for multiple interrelated goods, and on the other hand employs periodical, single-good, single-sided, sealed-bid, second-price auctions. The buyers are the user agents, the sellers are the owner agents, the market is an independent service, and the goods are CPU slots and disk capacities. The trading procedure is (1) the buyers bid for CPU and disk they wish to use, (2) the sellers ask for CPU or disk they can provide, (3) the market determines the winners either by the commodities market or by the auctions, and (4) the winning buyers execute their jobs on the winning sellers. Note that, in the auction model, there are separate auctioneers for each goods and thus the buyers have an exposure risk. For example, if a buyer wins a

CPU but doesn't a disk, he needs to wait for the next chance to bid on another disk while paying for the CPU. Because of this shortcoming the authors conclude that the commodities market model is more suitable for Grids than the auction model; however, this comparison is unfair because the auction model they employ is not a combinatorial auction but a traditional single-good auction. Additionally, the commodities market is typically unable to deal with arbitrary (not pre-defined) kinds of goods. Therefore the G-commerce's scheme does not directly fit the enterprise cloud environment.

Nimrod/G [25] by Buyya et al. is a negotiation-based Grid scheduler extended from Nimrod [26] built on the top of Globus Toolkit [27]. It aims to help researchers run a parameter survey program on heterogeneous resources while meeting the deadline and budget constraint. Nimrod/G employs a commodity market model with the parameters of time and cost. The buyer is an end-user, the seller is a resource provider (i.e. a computing node), the market is Nimrod/G components (specifically the scheduler under the control of the Parametric Engine), and the good is a right to use the resource. The trading procedure is (1) the buyer requests the scheduler via the Parametric Engine to arrange a resource that satisfies his deadline and cost, (2) the scheduler negotiates with the resource providers and selects one that meets the deadline at a minimum cost, and (3) the scheduler dispatches the user's job to the selected resource. Nimrod/G is specially designed for massively parallel applications on the scientific Grid environment, and thus it does not fit the enterprise cloud environment where several kinds of applications need to run in conjunction with others.

OCEAN [28] by Padala, et al. provides a market-based framework that enhances various middlewares for cluster and grid computing. It aims to be an infrastructure for high-performance computing environment where the resources are traded as commodities. OCEAN employs a hybrid model of tendering and

bargaining in a P2P network. The buyers and the sellers are represented by OCEAN Nodes, the matchmakers are implemented within OCEAN Nodes, and the good is a right to use resources specified by CPU/memory/disk/network/auxiliary hardware/ software/database/etc. The trading procedure is (1) a buyer propagates his trade proposal to nearby sellers, (2) a seller also propagates his trade proposal to nearby buyers, (3) the buyer starts negotiations by offering contracts to the potential sellers, (4) the seller accepts or rejects the contract and return it back to the buyer, (5) the buyer examines the returned contracts and signs the preferable ones, (6) the seller checks the signed contract and counter-sign it, and (7) the buyer starts using the resources. OCEAN's scheme does not fit the enterprise cloud environment because (1) its P2P-based structure does not provide a single marketplace where world-wide information on resource supply/demand are exchanged all together, and (2) its negotiation-based procedure does not guarantees neither an economically efficient allocation of resources nor an fair competition among the participants.

Bellagio by AuYoung et al. [29] is the first combinatorial auction-based marketplace for distributed computing resources. It aims to be a resource discovery and allocation system for distributed computing infrastructures such as PlanetLab [30]. Bellagio employs periodical, combinatorial, single-sided, sealed-bid, second-price auctions. The buyers are end-users, the seller is a computer site, the auctioneer is centralized Bellagio server, and the good is a right to use a resource discovered by SWORD [31]. The trading procedure is (1) the buyer discovers the sellers who satisfy his requirements, (2) the buyer bids for a bundle of resources like "20 of A and 30 of B at $5 or less", (3) the auctioneer determines the winners once an hour, and (4) the winning buyer uses the resources. Bellagio's scheme is well designed so that the buyer can express substitutability as well as complementarity between different kinds of resources. However, it does not fit the enterprise cloud environment because (1) the sellers are out of competition, and (2) the buyers

cannot bid for a workflow consists of multiple resources used on different timeslots.

Mirage by Chun et al. [32] is another combinatorial auction-based resource management system for SensorNet testbeds. It had been deployed in a real-world testbed and brought practical knowledge about the users' behavior. Mirage employs periodical, combinatorial, single-sided, sealed-bid, first-price auctions. The buyers are end-users, the seller is an agent on behalf of the sensors, the auctioneer is Mirage daemon running on front-end web server, and the good is access to the sensor. The trading procedure is (1) the buyer bids for sensors specified by abstract requirements, (2) the auctioneer translates the requirements into concrete set of sensors using resource discovery service, (3) the auctioneer determines the winners, and (4) the winning buyers get access to the sensors. Mirage's scheme is much similar to Bellagio's and it does not fit the enterprise cloud environment because of the same reasons as Bellagio.

Tycoon [33] by Lai et al. is a market-based distributed resource allocation system based on proportional share, where the resources are allocated in proportion to the amount of money the user spends. It aims to allow users to differentiate the value of their jobs in a cluster computing environment. Tycoon employs Auction Share algorithm to allocate resources instantly and reliably. The buyers are user agents, the sellers are hosts in the cluster, the auctioneer is a process running on the host, and the good is a right to use resources like CPU cycles. Note that there are multiple independent auctioneers on each host. The trading procedure is (1) the seller registers himself to the service location service, (2) the buyers bid on an auctioneer for the resource, (3) the auctioneer determines the quota on the resource for the buyers, and (4) the buyers use the resource. Tycoon's scheme does not fit the enterprise cloud environment because its distributed single-good auction model causes an exposure risk for the buyers who want a bundle of multiple resources.

CATNETS by Eymann et al. [34–37] compares the decentralized *Catallactic* approach with the centralized auction-based approach for resource allocation in Grid computing environment. Although the authors concluded that the decentralized approach fits to the Grid environment in terms of scalability, hereafter only the centralized approach is focused as it is the main concern of this thesis. The centralized approach employs periodical, combinatorial, double-sided, sealed-bid, K-pricing auctions. CATNETS divides the trading into two layers: a service market and a resource market. In the service market, the buyers are Complex Service Agents on behalf of the end-users, the sellers are Basic Service Agents, and the goods are services like PDF generation service. In the resource market, the buyers are Basic Service Agents, the sellers are Resource Service Agents on behalf of the owners, and the goods are resources like CPU/memory/storage/etc. In both markets, the auctioneer is an independent entity. The trading procedure is (1) a Resource Service Agent asks for his resource as seller in the resource market, (2) a Basic Service Agent asks for his service as seller in the service market, (3) a Complex Service Agent bids for a bundle of services as a buyer in the service market, (4) the auctioneer of the service market determines the winners, (5) the winning Basic Service Agent bids for a bundle of resources as a buyer in the resource market, (6) the auctioneer of the resource market determines the winners, (7) the winning Basic Service Agent uses the resources to provide his service to the winning Complex Service Agent, and (8) the winning Complex Service Agent uses the service. Unlike former literatures, CATNETS' scheme is much likely to fit the cloud computing environment as its Resource/Basic/Complex Services respectively correspond to the IaaS/PaaS/SaaS layers in the cloud. Nonetheless, the CATNETS' combinatorial auction model lacks the ability to deal with a workflow-oriented application which consists of multiple services running not at the same time.

SCDA by Tan et al. [38,39] proposes an iterative combinatorial exchange for

resource allocation in Grid computing environments, which essentially emulates a combinatorial auction by doing single-good auctions repeatedly. It aims to eliminate unnecessary volatility of the market price observed in conventional continuous double auctions. SCDA employs continuous, single-good, double-sided, sealed-bid, K-pricing auctions. The buyers are user agents, the seller is an owner agent, the auctioneer is an independent agent, and the good is an arbitrary kind of computing resource. The trading procedure is (1) the buyers bid for a certain amount of resource they wish to use, (2) the seller asks for his available resource, (3) the auctioneer determines the winners, and (4) the winning buyer uses the resource of the winning seller. SCDA's scheme does not fit the enterprise cloud environment because the buyers still have an exposure risk.

Table 2-3 summarizes these market-based resource management systems in literature for distributed computing environment.

Table 2-3 Summary of market-based computing resource management systems

Name	Year	Market Model	Centra-lized	Auction Design			
				Period-ical	Combi-natorial	Double-sided	Pricing
Spawn	1992	Auction	No	Yes	No	No	Second price
Popcorn	1998	Auction	Yes	Yes	No	No	Second price
						Yes	K-pricing
G-commerce	2000	Commodity market	Yes	–	–	–	–
		Auction	Yes	Yes	No	No	Second price
Nimrod/G	2002	Commodity market	No	–	–	–	–
OCEAN	2003	Tendering / Bargaining	No	–	–	–	–
Bellagio	2004	Auction	Yes	Yes	Yes	No	Second price
Mirage	2005	Auction	Yes	Yes	Yes	No	First price
Tycoon	2005	Proportional sharing	No	–	–	–	–
CATNETS	2005	Auction	Yes	Yes	Yes	Yes	K-pricing
SCDA	2007	Auction	Yes	No	No	Yes	K-pricing

2.3.2 In Production

As of early 2011, we can find some sort of cloud marketplace. An example in the IaaS layer is Amazon EC2 Spot Instances [40], where the provider dynamically sets the price of his IaaS and the users bid for it. The user's instance is shutdown if his bid undergo the price. Another example in the PaaS layer is Heroku Add-ons [41], where the provider sells his PaaSes at a posted price and the users buy them as a component of their applications. Further example in the SaaS layer is Google Apps Marketplace [42], where the providers sell their SaaSes at a posted price and the users compare and buy them to integrate into their business. At the writing point of this thesis, we noticed no double-sided auction system for cloud computing services in production.

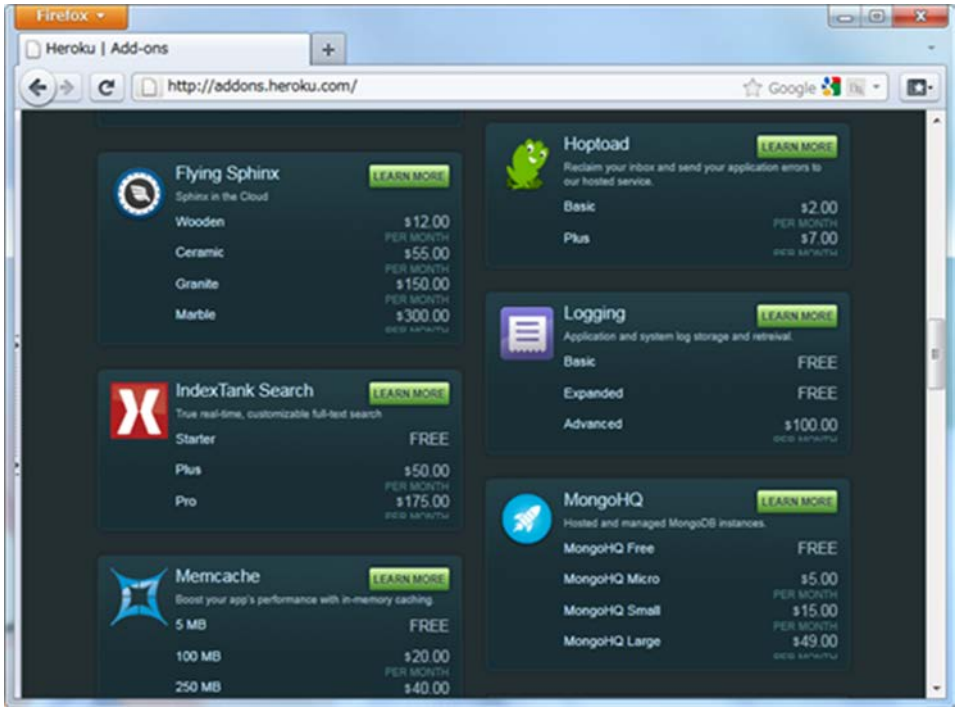


Figure 2.3 Heroku Add-ons

2.4 Other Disciplines

Market mechanism is widely used in the real world. This section give a glance to other use cases of market and auction mechanisms, while discussing the difference between cloud computing environments.

2.4.1 Electricity Exchange

Electricity markets are in practical operation for several years. For instance, Japan Electric Power Exchange (JPEX) started operations in 2005. According to ref. [43], it provides three markets: (1) a spot market for trading the electricity on the next day, (2) a forward market for trading the electricity to be delivered weeks or months ahead, and (3) a forward bulletin board market for free transactions. Since electricity and computing services have similar natures (i.e. they cannot be stored), the electricity market can be regarded as a preceding model to the cloud services market. However, the electricity market model cannot be directly applied to cloud computing because the electricity is almost uniform, whereas computing services vary in type and quality.

2.4.2 Stock Exchange

The stock market deals with a variety of stocks, which can be stored and resold, unlike a computing service. The studies on dealing strategies and mechanism design have used multi-agent simulations. U-Mart [44] is a test bed for multi-agent simulations of the stock market, and it is especially focused on futures trading. It allows machine agents and human agents to trade future stocks at the same time. Our simulator presented in this thesis is developed with an interface with the U-Mart system, so that we can evaluate performance of various machine/human agents.

2.5 Summary

This chapter has extensively reviewed the market mechanisms from four points of view: (1) historical economics, (2) auction theory, (3) computer science, and (4) other applications in our society. Now we know that the market mechanism has been widely studied by the economists and by the computer scientists. However, there exists no market mechanism that just fits today's cloud computing environment, because the cloud computing model is a new idea having no similarities in the past. Particularly, the combinatorial, workflow-oriented nature of cloud-based enterprise applications makes it impossible for any existing market mechanisms to deal with such a combination of services as a bundle. This thesis therefore proposes a brand new design of marketplace that realizes trading of any combination of services with different timeslot as a bundle, as well as fair competition and economically efficient outcome.

Chapter 3 Cloud Computing Model

3.1 Cloud Service

The "cloud" spreads over a wide level of abstraction from hardware to software. Now it is understood in a three-tier structure from low-level to high-level: Infrastructure as a service (IaaS), Platform as a service (PaaS) and Software as a service (SaaS). Many providers compete in each tier for selling their own services, making it increasingly difficult for users to select an appropriate service among them.

In this thesis, an enterprise system is assumed to be implemented using PaaS. PaaS is becoming a major methodology to build an enterprise system on it, because the developer can build the system without procuring and configuring hardware/software; thus, the customer can significantly reduce cost and time for development. The number of PaaS provider is expected to increase as the platform technology is becoming standardized or open-source software is available [45–54].

The price of PaaS is assumed to be set on a per-process-per-hour basis for each type of service in this thesis. This assumption is reasonable to simulate the existing PaaS model. For example, Heroku [54] charges for dynos, workers, databases and add-ons separately. The dyno is a front-end process responsible to HTTP requests ("more dynos provide more concurrency") and the worker is a back-end process

responsible for queued jobs ("more workers provide more capacity"), both of which costs $0.05 per process per hour; while the database costs monthly depending on its performance.

3.2 Enterprise System

An enterprise system generally consists of multiple subsystems running in parallel and/or sequentially, each of which requires a guaranteed quality of service (QoS) at a predictable price. Each enterprise system, or each user's request, is assumed to be represented by workflow. An example of business workflow is a payroll system [55]. It consists of a payroll calculation task on Java service along with an employee database task on SQL service, followed by reporting task on PDF/Email service as shown in Figure 3.1. Another example of engineering workflow is a CAE system [56]. It consists of a mesh generation task and a CFD analysis task on a HPC service, controlled by an optimization task on a general-purpose optimization service. Every task needs to reserve the specified type of service within an

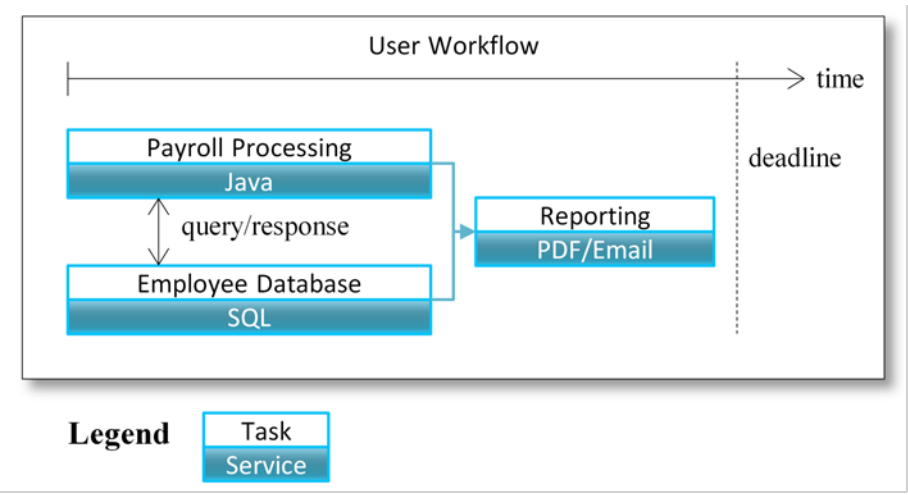


Figure 3.1 Enterprise system model

appropriate timeslots to meet a deadline. The overall cost should also be restricted by the user's total budget. Each task in the workflow is implemented using PaaS; thus, the user needs combination of PaaS services to organize the user's workflow.

Figure 3.2 illustrates another story of a company depends on SaaSes to run their business. The sales division uses a supply chain management service at the midnight to aggregate their daily records. The investment division uses a risk assessment service for several hours at the weekend to evaluate their assets. The accounting division uses a human resource management service for three days at the end of month to calculate the payroll. The design division uses a computer aided engineering service to meet a deadline of a product. The research division has their own computers currently not in use; so they can host other applications and earn from its user. The former three divisions need scheduled allocation of services for regular tasks, whereas the latter two divisions need immediate allocation of services

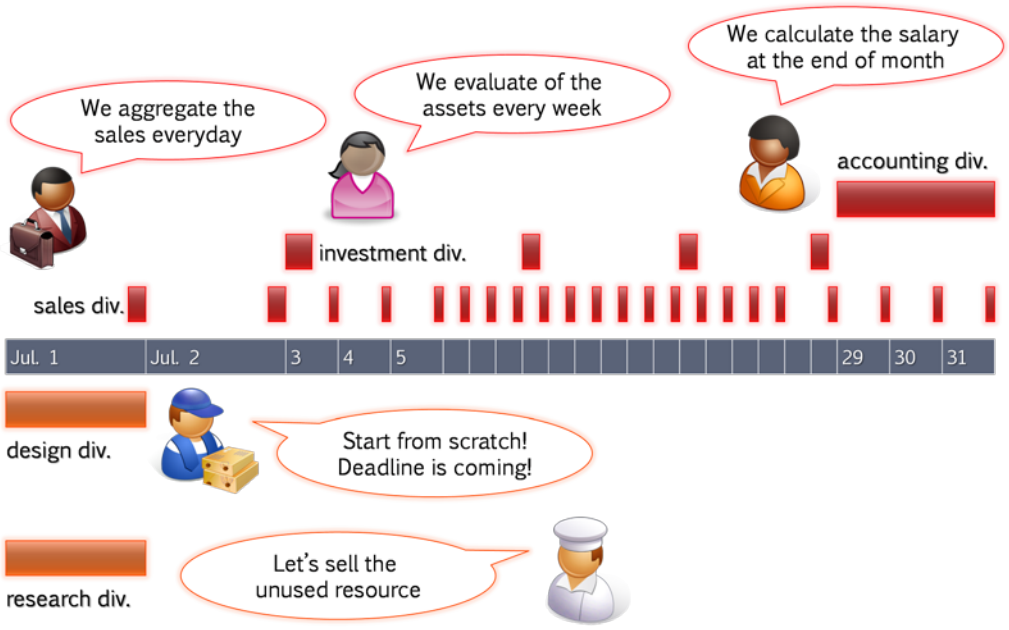


Figure 3.2 Example SaaS usage

for irregular tasks. Consequently, the marketplace should support these scheduled and immediate allocations to meet the requirements of enterprise usage.

3.3 Requirements

The service allocation decided by the marketplace must be fair and efficient; otherwise the user will have no incentive to take part in. Hence, the PaaS marketplace is assumed to have to support the following requirements:

- *Combination of services*

  Each user needs to bundle multiple services with different start/finish times as mentioned above. The cloud marketplace should allow users to express complementary requirements for an arbitrary combination of services.

- *Predictability and flexibility*

  Since supply and demand in the cloud computing environment changes dynamically over time, users may desire predictable allocation in advance and adjustment at runtime.

- *Economic efficiency*

  Every user and provider desires a fair and efficient allocation of services. The cloud marketplace should maximize the benefit of the participants and should not waste any resource. To this end, it is preferable for the marketplace to adopt an exact optimization approach rather than a heuristic approach.

- *Double-sided competition*

  To encourage a fair exchange between providers and users, the prices should only depend on supply-demand condition, giving no structural advantage on a seller's (provider's) side or a buyer's (user's) side. The cloud marketplace should be designed after the double-sided auction model, meaning that the providers and the users compete with each other.

3.4 Marketplace Model

Figure 3.3 illustrates an ideal marketplace model that fits to the enterprise usage and requirements mentioned above. It should support multiple types of services, an application composed of multiple services as a workflow or as co-allocation, price bidding by both providers and users, and a fair outcome satisfying economic efficiency. The next chapter describes the proposed market model in detail.

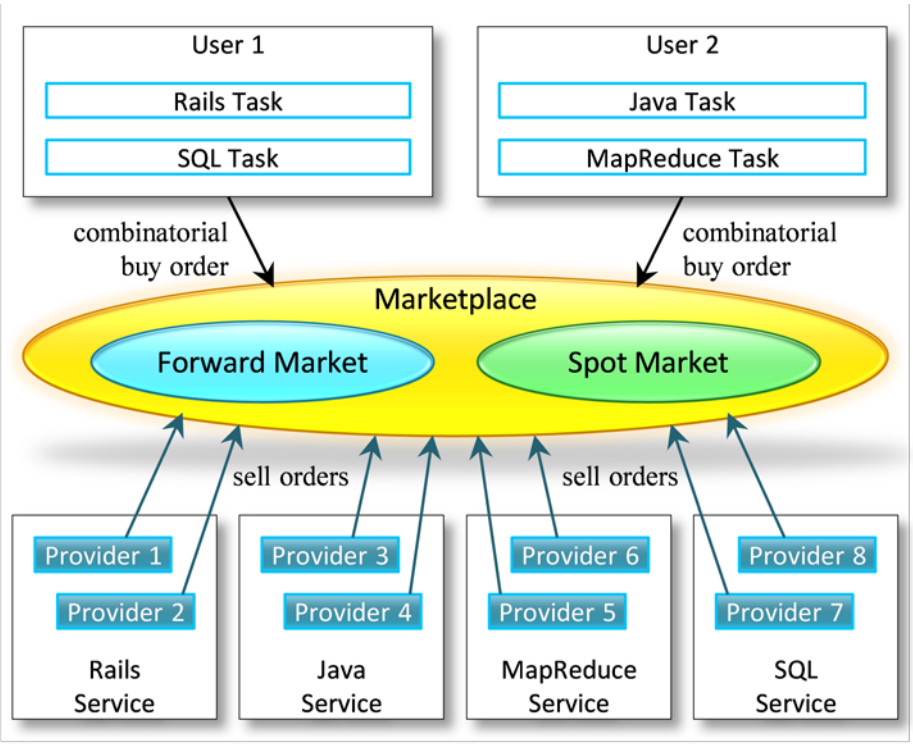


Figure 3.3 Marketplace model

(This page is intentionally left blank)

Chapter 4 Market Mechanism

4.1 Overview

Figure 4.1 illustrates an overall perspective of the cloud computing environment with the proposed marketplace mechanism. The marketplace has two independent markets: the spot market for a short-term reservation (e.g. in one hour) and the forward market for a long-term reservation (e.g. in one month). The service

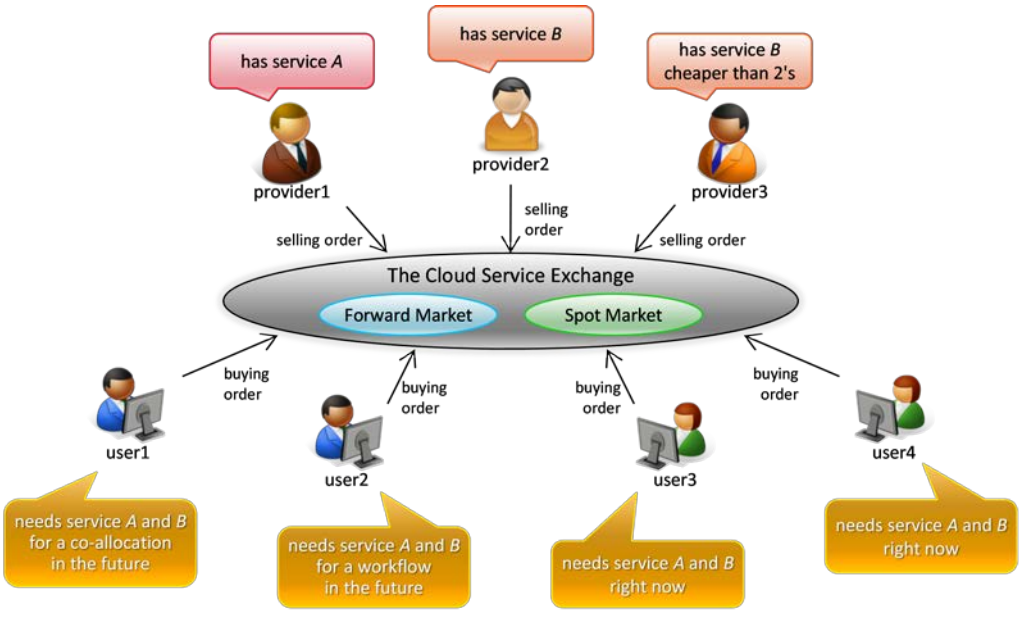


Figure 4.1 Overview of the proposed marketplace

providers participate in the markets as sellers while the users participate as buyers. For instance, the provider places sell orders with a market when he has a capacity of service with specific QoS. The user places a buy order with a market when he builds an application using specific services. The market accepts sell/buy orders for fixed duration, and then it determines the allocation of the services between the providers and the users. Finally, the market informs the participants of the resulting allocation to allow the user to deploy his application and the provider to preserve his capacity. We refer to the time of actual usage/provision as "delivery" and the sequence of the above procedures as a "round". The marketplace repeats the rounds periodically.

4.2 Hypothesis

A service and a task are assumed to satisfy the following conditions as shown in Figure 4.2:

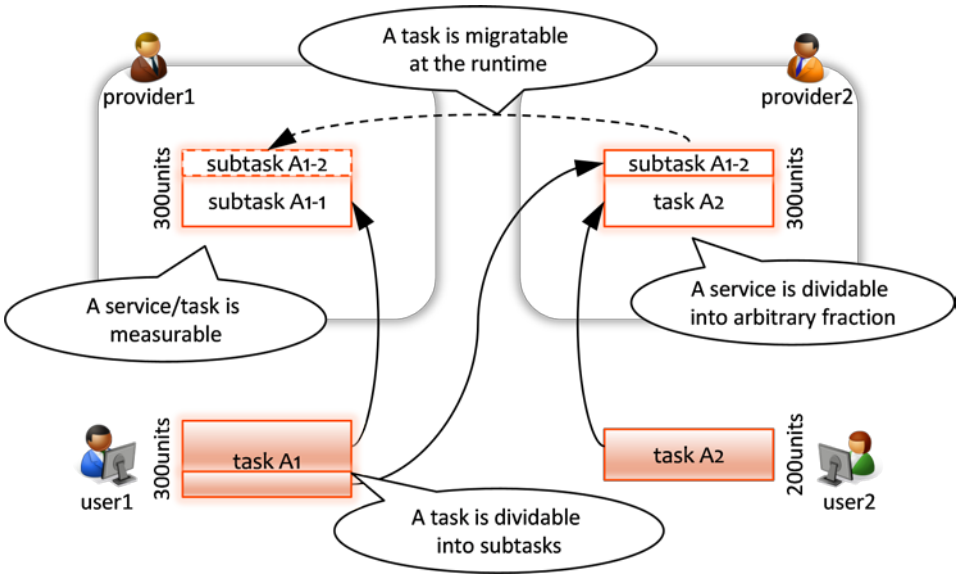


Figure 4.2 Service and task

An amount of resources that satisfy QoS is represented as a one-dimensional value, e.g. the number of processes or the performance of virtual machine. We refer to the amount of resources as "quantity" and represent it with the metric "units" in the rest of this thesis. The price of resources is assumed to be proportional to its quantity.

A provider can host multiple users at the same time unless exceeding its capacity. For instance, the provider can provide 20 units to a user and 40 units for another user when it has a capacity of 60 units.

A user can build an application using services offered by multiple providers to fulfill his demand. For instance, the user uses 10 units on the provider 1 and 30 units on the provider 2 when he builds an application consuming 40 units.

An application can be migrated at runtime, i.e. an application running on a provider can be suspended, moved and resumed on another provider.

The physical parameters, such as network bandwidth and migration time, are omitted in this model for the sake of simplicity. The physical cost can be included in a price or traded as a separate service in reality.

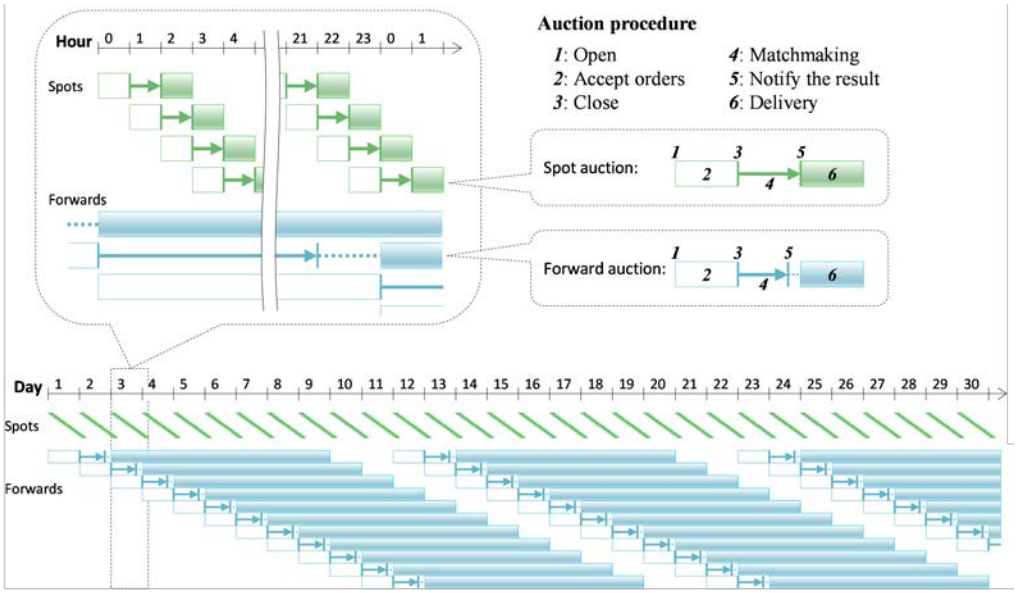


Figure 4.3 Trading schedule

4.3 *Trading Schedule*

Each of the two markets holds clearinghouse auctions periodically. Figure 4.3 shows the schedule of the auctions. The spot and the forward auctions have the same procedure in different timescale. Here sellers and buyers are treated equally as participants.

In the spot market, a participant willing to sell/buy services in one-hour timeslot t needs to submit a request during the timeslot $t-2$, which begins two hours prior and ends one hour prior to the requested timeslot. For instance, let us suppose trading services at the third timeslot (from 2:00 to 3:00). The market opens for one hour at the first timeslot (from 0:00 to 1:00) to accept sell/buy orders from the participants. The market then closes and starts matchmaking, i.e. computes the optimal allocation of services from providers to users. Within one hour (by 2:00) the matchmaking finishes and the market notifies the results to the participants. Finally

the participants utilize the services they win during the third timeslot (from 2:00 to 3:00).

Trading in the forward market is same as the spot market except the timescale of the procedure. Let F denotes the length of the forward delivery days[2] indicated by the blue-colored boxes marked 'forwards' in Figure 4.3. A participant willing to sell/buy services starting on the f-th day needs to submit a request during the days between $f-(F-1)$ and $f-2$, that begins $F-1$ days prior and ends two days prior to the delivery day. For instance, the market opens a forward auction for one day (from 0:00 to 24:00) on the 1st day. Assuming $F=7$ the participants can place orders for services delivered between 0:00 on the 3rd day and 24:00 on the 9th day. The forward market performs matchmaking and notifies the results advising when to utilize the services. Note that the forward matchmaking can spend at most 22 hours; this is the design to support time-consuming MIP technique. Since the spot market opens at 22:00 for the next day's services, the forward matchmaking must finish before 22:00.

[2] (financial term) The day on which seller/buyer actually provide/use the services

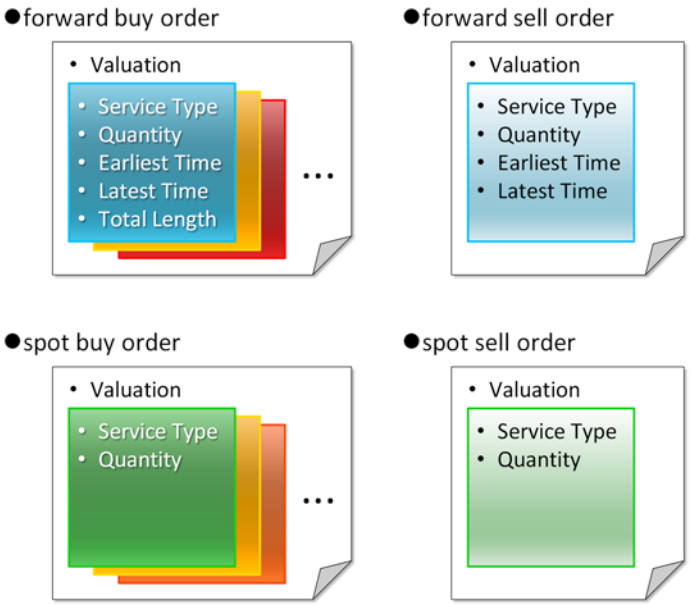


Figure 4.4 Bidding language

4.4 Bidding Language

The bidding language determines the information included in an order. Figure 4.4 illustrates the order forms sent from a participant to a market.

A buy order from a user has a valuation (maximum price he wishes to pay) and a bundle of arbitrary services he needs. For each service the user specifies a service type, quantity, the earliest timeslot acceptable to start (arrival time of the task), the latest timeslot acceptable to finish (deadline of the task) and the total number of timeslots (estimated runtime of the task). The latter three parameters are only used in the forward market. Note that the valuation is given to a bundle of services, not to each discrete service, so that the user can express requirements for receiving multiple services in combination. If the market cannot reserve all the services in a bundle at once, the user receives nothing at all.

A sell order from a provider has a valuation (minimum price he wishes to earn) and a service he offers. The provider specifies a service type, quantity, the earliest timeslot and the latest timeslot available for use. The latter two parameters are only used in the forward market. Note that a sell order includes only one service. The provider can make separate orders for different services. If a provider wishes to sell certain low-level services at the same time, he can do so by bundling them into a single high-level service.

Formulation: Let $M = \{m_1, \dots, m_{|M|}\}$, $m_i = \{v_i, O_i\}$ be sell orders; $N = \{n_1, \dots, n_{|N|}\}$, $n_j = \{v_j, O_j\}$ be buy orders; $G = \{g_1, \dots, g_{|G|}\}$ be service types; $1 \le t \le T$ be timeslots; and v_i and v_j be valuation. A buy order is formulated as

$$O_j = \{(g_k, q_{j,k}, b_{j,k}, d_{j,k}, l_{j,k}) \mid 1 \le k \le |G|\}$$

where $q_{j,k}$ is the quantity of service g_k, $b_{j,k}$ is the earliest beginning timeslot, $d_{j,k}$ is the latest ending timeslot and $l_{j,k}$ is the number of timeslots[3]. Similarly, a sell order is formulated as

$$O_i = (g_k, q_{i,k}, b_{i,k}, d_{i,k}), \qquad 1 \le k \le |G|.$$

where O_i is a vector. Again, note that b, d and l are only valid in forward orders.

4.5 Allocation Scheme

The allocation scheme determines the winners of an auction, or allocation of services from providers to users. Our goal is to get an economically efficient (or Pareto optimal) allocation of resources, where it is impossible to increase a participant's surplus without decreasing another participant's surplus. Here the surplus means the difference between the market price and the participant's internal

[3] If $l_{j,k} < d_{j,k} - b_{j,k} + 1$ then the task k may be suspended/resumed during runtime.

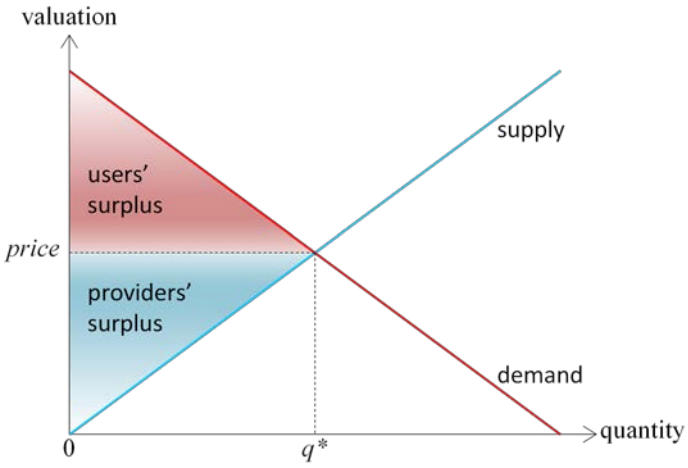


Figure 4.5 Illustration of surplus and welfare

valuation; i.e. (price – cost) for the provider and (utility – price) for the user, as shown in Figure 4.5. The aggregate surplus, i.e. the difference between the buyers' valuation and the sellers' valuation, is also known as the social economic welfare.

Maximizing the social economic welfare w is the sufficient condition for Pareto optimality [57]. Therefore we formulate the winner determination problem into a linear mixed integer program (MIP) and try to exactly maximize w. Here, four decision variables are introduced: $u_j \in \{0,1\}$ denotes whether the buyer n_j gets all services in the bundle; $x_{j,k} \in \{0,1\}$ denotes whether the service g_k is allocated to the buyer n_j; $z_{j,k,t} \in \{0,1\}$ denotes whether the service g_k is allocated to the buyer n_j in the timeslot t; $0 \leq y_{i,j,k,t} \leq 1$ denotes the percentage of the service allocated to the buyer n_j in the timeslot t, where the service g_k is owned by the seller m_i. The solver then maximizes the social economic welfare w by solving the MIP:

Maximize

$$w = \sum_{j \in N} v_j u_j \; - \sum_{i \in M} \sum_{j \in N} \sum_{k \in G} \sum_{t \in T} v_i y_{i,j,k,t} \tag{1}$$

s.t.

$$\sum_{k \in G} x_{j,k} - \; |G|\, u_j = 0, \qquad j \in N \tag{2}$$

$$\sum_{t \in T} z_{j,k,t} - l_{j,k}\, x_{j,k} = 0, \qquad j \in N, k \in G \tag{3}$$

$$\sum_{j \in N} y_{i,j,k,t} \leq 1, \qquad i \in M, k \in G, t \in T \tag{4}$$

$$q_{j,k} z_{j,k,t} - \sum_{i \in M} q_{i,k} y_{i,j,k,t} = 0, \qquad j \in N, k \in G, t \in T \tag{5}$$

$$\left(b_{j,k} - t\right) z_{j,k,t} \leq 0, \qquad j \in N, k \in G, t \in T \tag{6}$$

$$\left(t - d_{j,k}\right) z_{j,k,t} \leq 0, \qquad j \in N, k \in G, t \in T \tag{7}$$

$$\left(b_{i,k} - t\right) \sum_{j \in N} y_{i,j,k,t} \leq 0, \qquad i \in M, k \in G, t \in T \tag{8}$$

$$\left(t - d_{i,k}\right) \sum_{j \in N} y_{i,j,k,t} \leq 0, \qquad i \in M, k \in G, t \in T \tag{9}$$

$$u_j \in \{0,1\}, \qquad j \in N \tag{10}$$

$$x_{j,k} \in \{0,1\}, \qquad j \in N, k \in G \tag{11}$$

$$z_{j,k,t} \in \{0,1\}, \qquad j \in N, k \in G, t \in T \tag{12}$$

$$0 \leq y_{i,j,k,t} \leq 1, \qquad i \in M, j \in N, k \in G, t \in T \tag{13}$$

4.6 Pricing Scheme

A price earned/paid by a provider/user for an allocation is decided by the pricing scheme. The pricing scheme should be budget balanced and individually rational in order to sustain the market and give providers/users incentives to participate in the market. The former means that total earnings of providers is equal to the total payment of users, and the latter means a provider/user earns/pays no less/more than their valuation.

The K-pricing scheme [58] is employed to meet the above requirements. The basic idea of K-pricing is to distribute the social economic welfare among the users and the providers. It is straightforward in non-combinatorial auctions. In our combinational auctions, however, we can neither calculate the discrete price for each service nor for each timeslot of a user's order. Here, the following algorithm is introduced to determine the price.

Assume $u_j = 1$, since the only orders that succeed need pricing. Let $0 \leq K \leq 1$ be an arbitrary fraction. For a buy order n_j, let w_j be the welfare corresponding to n_j, p_j be the price, $p_{i,j}$ be the price earned by the provider i, and $r_{i,j,k,t}$ be the proportion of the provider i 's valuation to all the providers' valuation of the service k in timeslot t. They are formulated as

$$w_j = v_j - \sum_{i \in M} \sum_{k \in G} \sum_{t \in T} v_i y_{i,j,k,t} \tag{14}$$

$$p_j = v_j - (1 - K) w_j \tag{15}$$

$$r_{i,j,k,t} = \frac{\sum_{k \in G} \sum_{t \in T} v_i y_{i,j,k,t}}{\sum_{i \in M} \sum_{k \in G} \sum_{t \in T} v_i y_{i,j,k,t}} \tag{16}$$

$$p_{i,j} = \sum_{k \in G} \sum_{t \in T} v_i y_{i,j,k,t} + K \sum_{k \in G} \sum_{t \in T} w_j r_{i,j,k,t} \tag{17}$$

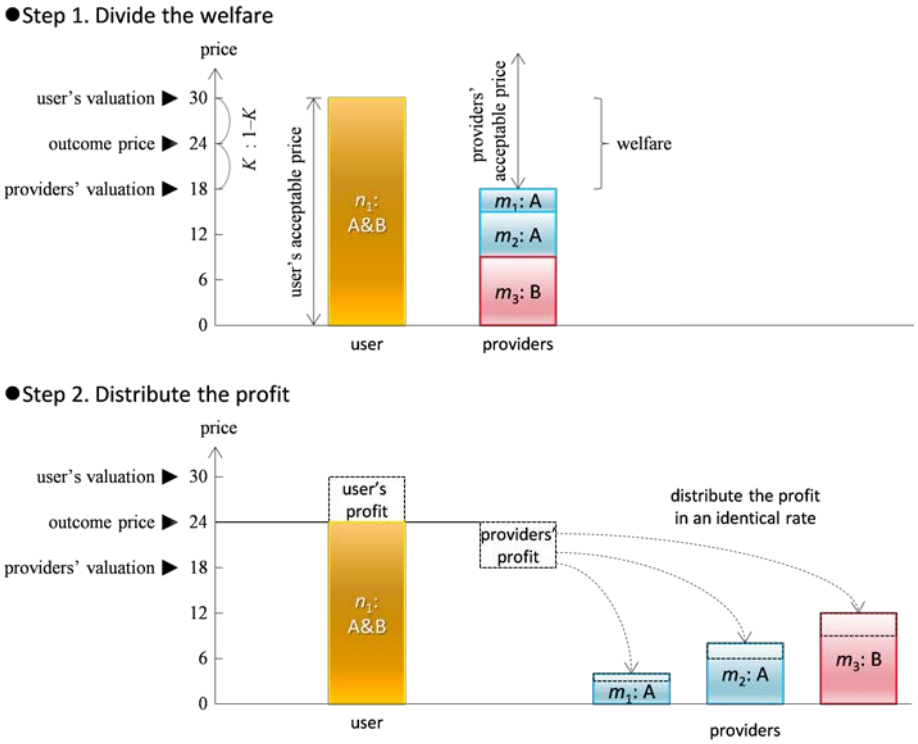


Figure 4.6 Pricing scheme

Consequently, the provider i's total earning p_i is formulated as

$$p_i = \sum_{j\in N}\sum_{k\in G}\sum_{t\in T} v_i y_{i,j,k,t} + K\sum_{j\in N}\sum_{k\in G}\sum_{t\in T} w_j r_{i,j,k,t} \qquad (18)$$

Let's see an example shown in Figure 4.6. A user n_1 ordered a combination of service A and B with a valuation of (at most)

$$v_{n_1} = \$30\,.$$

As a result of matchmaking, providers m_1, m_2 (both provides service A) and m_3 (provides service B) are allocated, and their valuations are (at least)

$$\sum_{k\in G}\sum_{t\in T} v_{m_1} y_{m_1,n_1,k,t} = \$3\,,$$

$$\sum_{k\in G}\sum_{t\in T} v_{m_2} y_{m_2,n_1,k,t} = \$6\,,$$
$$\sum_{k\in G}\sum_{t\in T} v_{m_3} y_{m_3,n_1,k,t} = \$9\,,$$

respectively. In this case, the welfare is

$$w_{n_1} = \$30 - (\$3 + \$6 + \$9) = \$12\,.$$

Assuming $K = 0.5$, the K-pricing scheme calculates the outcome price as

$$p_{n_1} = \$30 - (1 - K) \times \$12 = \$24\,.$$

This becomes the trading price for the user; he profits $\$6$ by the trading. For the providers K-pricing scheme calculates the distribution of profit as

$$\sum_{k\in G}\sum_{t\in T} r_{m_1,n_1,k,t} = \frac{\$3}{(\$3 + \$6 + \$9)} = \frac{1}{6},$$
$$\sum_{k\in G}\sum_{t\in T} r_{m_2,n_1,k,t} = \frac{\$6}{(\$3 + \$6 + \$9)} = \frac{1}{3}\,,$$
$$\sum_{k\in G}\sum_{t\in T} r_{m_3,n_1,k,t} = \frac{\$9}{(\$3 + \$6 + \$9)} = \frac{1}{2}\,.$$

So the trading prices for providers become

$$p_{m_1,n_1} = \$3 + K \times \$12 \times \frac{1}{6} = \$4\,,$$
$$p_{m_2,n_1} = \$6 + K \times \$12 \times \frac{1}{3} = \$8\,,$$
$$p_{m_3,n_1} = \$9 + K \times \$12 \times \frac{1}{2} = \$12\,,$$

respectively; they profit $\$1$, $\$2$ and $\$3$ by the trading.

The incentive compatibility, which means that the participant's dominant strategy is to reveal his valuation truthfully, is another important aspect of the pricing scheme. However, these three aspects —the budget balance, the individual rationality, and the incentive compatibility— cannot be fulfilled at the same time [59]. This thesis focus on the first two aspects, the budget balance and the individual

rationality, because non-truthful bidding should also be allowed as the participant's strategy.

(This page is intentionally left blank)

Chapter 5 Simulator

5.1 Overview

A Java-based simulator, named W-Mart, has been developed to explore market behavior by means of multi-agent simulations. The overall architecture of W-Mart is illustrated in Figure 5.1.

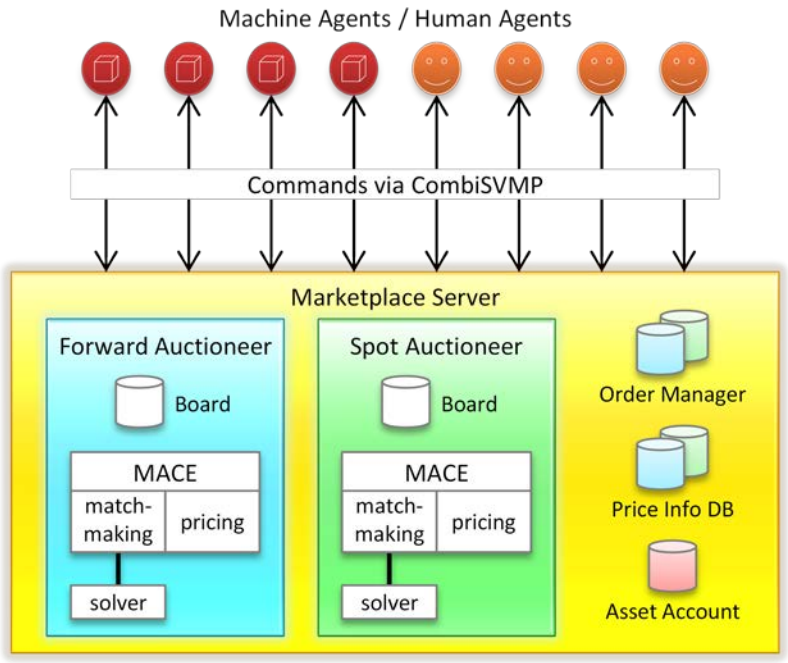


Figure 5.1 Overview of W-Mart system

The W-Mart system consists of a marketplace server, two auctioneers for the spot and forward markets inside the marketplace, and a number of participant agents outside the marketplace. The agent is either a machine agent (i.e. autonomous client software) or a human agent (i.e. an operator with terminal software). The marketplace is designed to deal with both types of agents simultaneously. The agents and the marketplace communicate by a dedicated protocol named CombiSVMP. All the components of W-Mart system are built on Java SE platform. The following sections describes the detailed design and implementation.

5.2 Participant Agents

The participant agents simulate the provider and the user of cloud computing services. An unlimited variation of agents can be developed and used with W-Mart as far as they talk CombiSVMP. This section introduces our own agents used for the experiments of this thesis. Although W-Mart allows human agents to participate in trading, they are not used in the experiments of this thesis. We thus focus on our machine agents and describe the design and implementation of them.

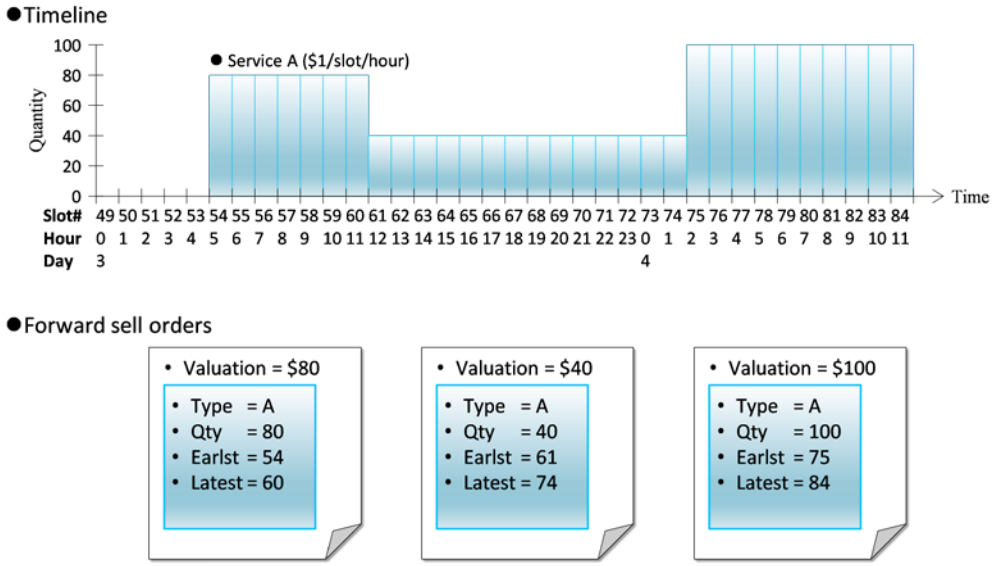


Figure 5.2 Example seller's timeline and orders

5.2.1 *Seller Agent*

A seller agent stands for a provider of cloud computing services. It is reasonable to assume that the provided quantity and valuation of services are constant, since these services are usually hosted on computational resources in huge datacenters. As such the seller agent implements a timeline to manage his "stock" resources and attempts to sell the "remainder" resources at a constant order price. We assume that one seller provides one service for the sake of simplicity.

Figure 5.2 depicts an example snapshot of a seller's timeline and his next order to be placed. His resource is sold out until slot#53 (hour 4 of day 3); after that the stock quantity is 80 units between slot#54-60, 40 units between slot#61-74 and 100 units thereafter. He then generates three sell orders according to the quantities and sends them to the forward auctioneer. He also generates orders to the spot auctioneer as well, excluding time-related properties.

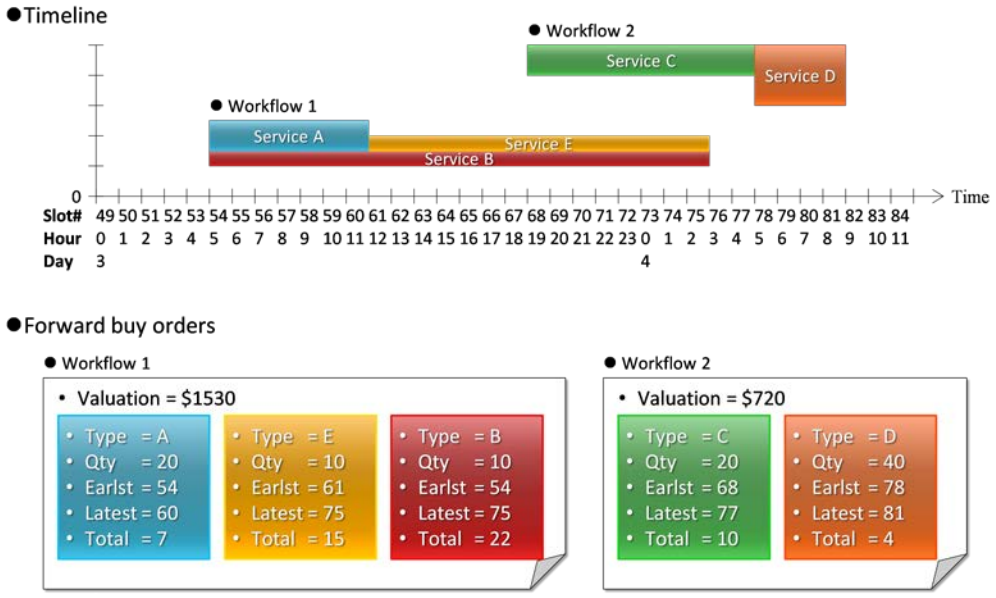


Figure 5.3 Example buyer's timeline and orders

5.2.2 Buyer Agent

A buyer agent stands for a user of cloud computing services. As the user's demand changes dynamically in realistic situations, our buyer agent is designed to follow pre-generated demands (i.e. a schedule of workflows). As such the buyer agent implements a timeline to manage his demands and attempts to buy all the services composing the workflows at the specified (also pre-generated) order price.

Figure 5.3 depicts an example snapshot of a buyer's timeline and his next order to be placed. He has two workflows to be executed: one starts with service A at slot#54 (hour 5 of day 3) followed by service E then finishes at slot#75, and both co-allocated with service B; another starts with service C at slot#68 and followed by service D then finishes at slot#81. He then generates two combinatorial buy orders correspondingly and sends them to the forward auctioneer. He also generates orders to the spot auctioneer as well, excluding time-related properties.

```
3/23,   8/11,   2618,   A:26:180:181:2;D:38:180:181:2;C:54:182:186:5;
1/23,   8/12,   199,    B:10:181:181:1;E:40:182:182:1;
6/23,   8/13,   4665,   A:27:182:184:3;B:90:182:184:3;E:50:182:184:3;D:29:185:185:1;
2/23,   9/15,   3097,   B:61:208:211:4;D:28:208:211:4;A:32:212:212:1;
4/23,   9/22,   442,    A:51:215:216:2;D:20:217:220:4;
7/23,   10/20,  571,    A:46:237:237:1;B:52:238:238:1;
```

Figure 5.4 Example scenario file

5.2.3 Scenario Generator

The schedule of workflows fed to the buyer agent is generated by scenario generator and stored in a CSV file. Figure 5.4 shows an example of it including six workflows. One line represents one workflow. The first column is the scheduled timeslot to send the order for the workflow, e.g. "3/23" represents "the workflow should be ordered at hour 23 of day 3". The second column is the starting timeslot of the workflow, e.g. "8/11" represents "the workflow starts at hour 11 of day 8". The third column is the order price (valuation) of the whole workflow. The fourth column is the OrderSpec of the workflow. The semicolon-separated chunks of an OrderSpec represent the services of the workflow. The colon-separated values of a chunk represent the parameters of the service: i.e. type, quantity, earliest slot#, latest slot# and total length, respectively. Our buyer agent reads this file on the beginning of simulation and builds his own timeline accordingly.

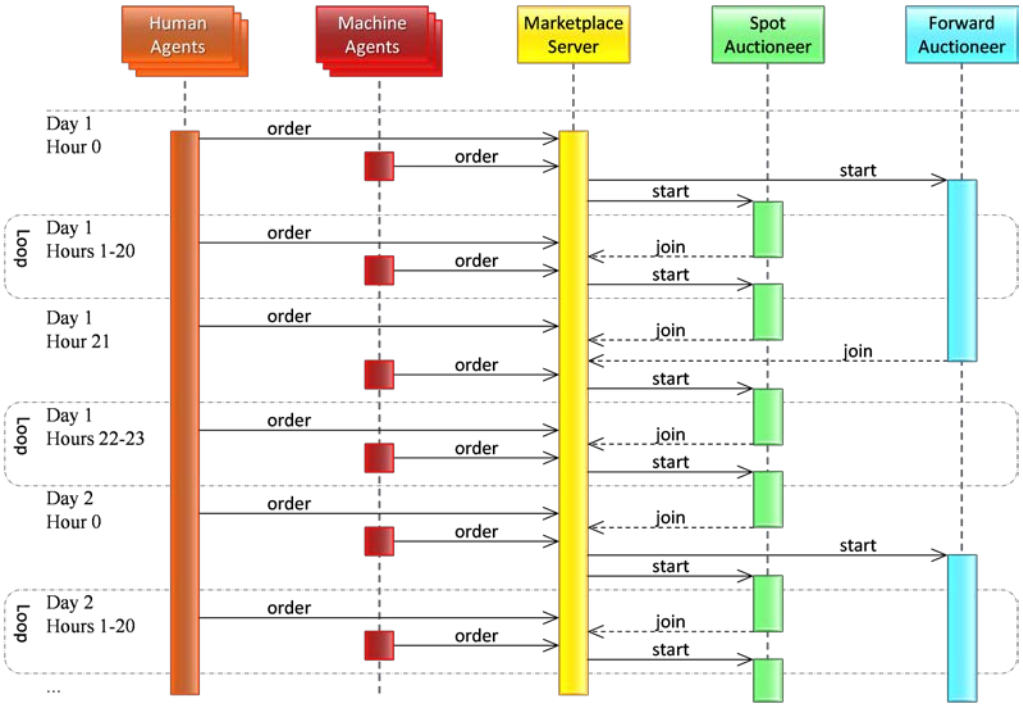


Figure 5.5 Digest sequence diagram of W-Mart system

5.3 Marketplace Server

The marketplace server orchestrates the overall system of W-Mart. Its key role is to manage the agents and the auctioneers. Timing of interactions with these entities is illustrated in Figure 5.5 (digest version) and Figure 5.6 (detailed version). Here the first day of the marketplace starts with accepting orders from human agents and machine agents. After receiving all orders the marketplace kicks off two auctioneers to execute auctions. Here the first hour ends and the next hour begins. The marketplace keeps open for human orders while waiting for the spot auctioneer to finish. After some ten minutes past, the marketplace closes for human orders and joins the spot auctioneer. Having the result of the spot auction, the marketplace then calls machine agents to make orders. After all the machine agents place their

orders, the marketplace again kicks off the spot auctioneer. This is the one round of spot trading and the marketplace repeats it until 21 o'clock. Note that the trading schedule of the former diagram corresponds to that of Figure 4.3, where the forward auction starts in 0 (zero) o'clock and finishes by 22 o'clock every day. So in the round of 21 o'clock, the marketplace joins the forward auctioneer in addition to the spot auctioneer. Here the results of the forward auction include the service allocation on the next day. The marketplace allows agents to know the tomorrow's reservation results and decide the orders on the next spot auction.

Another important role of the marketplace is to manage/keep/translate the orders and the outcome. As illustrated in Figure 5.1, the marketplace has an OrderManager to keep the orders and a PriceInfoDB to keep the market information. Once the marketplace receives an order by the OrderRequest command of CombiSVMP, it is translated into an Order object and is kept in the OrderManager. When the auctioneer finishes, the outcome and the market information are again translated into an Order object and a ContractInformation object and are kept in the OrderManager and the PriceInfoDB, so that the agents can acquire these information via the OrderStatus command and the MarketPrice command of CombiSVMP.

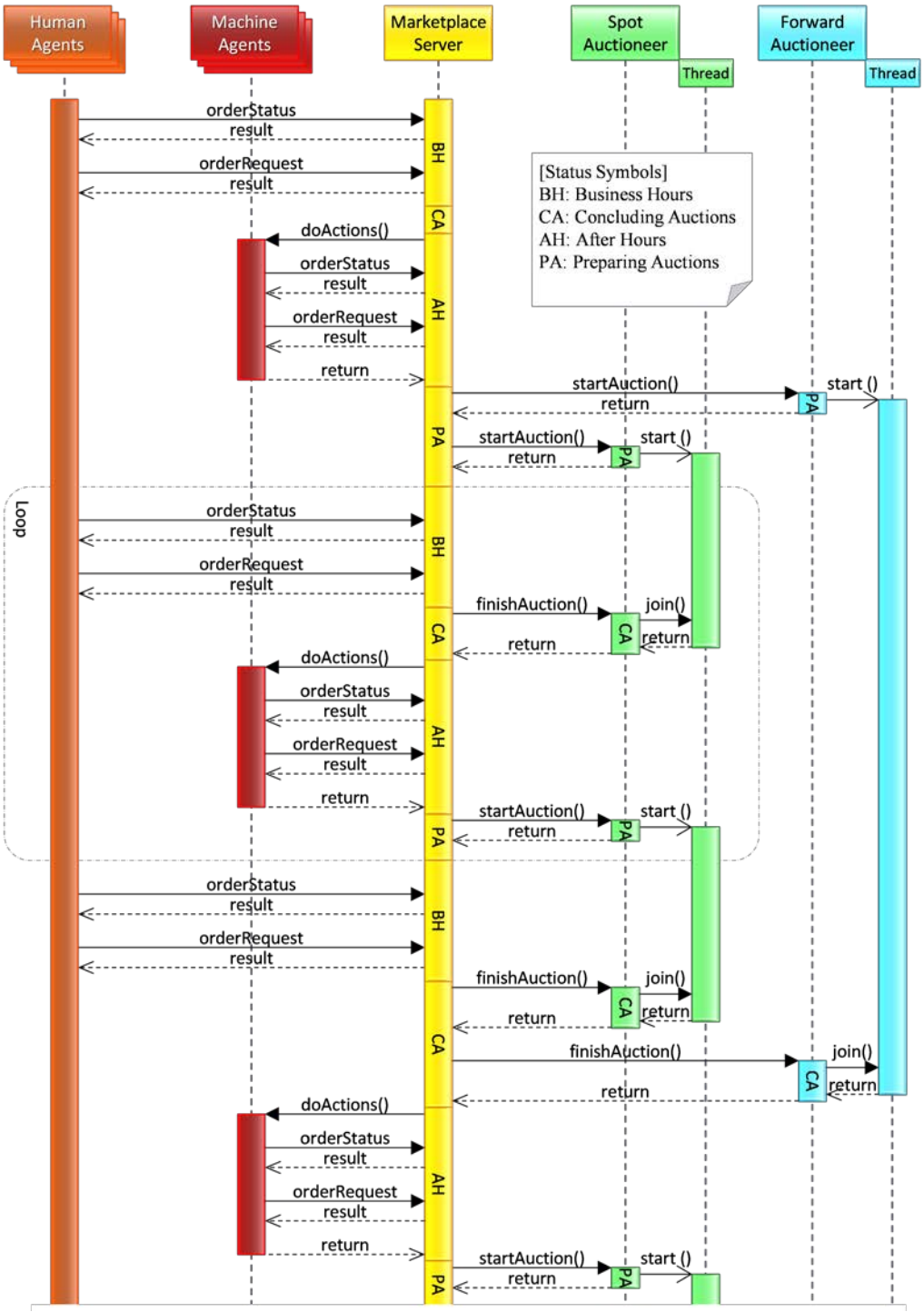


Figure 5.6 Detailed sequense diagram of W-Mart system

5.4 Auctioneers

There are two auctioneers within the marketplace server, namely the forward auctioneer and the spot auctioneer, standing for the forward market and the spot market, respectively. As the spot auctioneer is a special case of the forward auctioneer where the time-related parameters are omitted, we focus on the forward auctioneer and describe its design and implementation in this section.

Figure 5.6 illustrates that the auctioneer has its own thread to run asynchronously with the marketplace server. After the agents finish placing their orders, the marketplace server moves all orders (including uncontracted orders of previous session) from the OrderManager to the Board of the auctioneer as shown in Figure 5.1, and then he starts the auctioneer's thread. Once started, the auctioneer translates the orders from the Board into the MACE [60] framework, and kicks it off to execute matchmaking and pricing. When the MACE framework completed the calculation, the auctioneer scans the outcome and writes it back to the Board. On the other hand, the marketplace server goes asynchronously to the next session, and waits for the auctioneer's thread to terminate. Here the marketplace server may be waiting for the thread to finish. After that the marketplace server moves the outcome from the Board to the OrderManager, as well as the market information (such as the market price and volume) from the Board to the PriceInfoDB. Finally the outcome and the market information can be retrieved by the agents via the OrderStatus and the MarketPrice commands, described in the following section.

The MACE framework embedded in the auctioneer plays two main roles: (1) a driver of combinatorial auction and (2) an executer of pricing scheme. The WinnerDetermination component of MACE is responsible for computing an allocation, i.e. it implements the mixed integer program. After the computation of the allocation finishes, the prices are computed by the Pricing component of MACE.

These components are pluggable and are easy to be exchanged. We developed our own versions of them for W-Mart, implementing the proposed mechanism described in Chapter 4.

The bottommost layer of MACE comprises of third party components including an MIP solver. The current implementation of MACE employs JOpt [61] as a wrapper to support multiple MIP engines, namely CPLEX [62] and lp_solve [63]. CPLEX is the state-of-the-art optimization engine for mixed integer programs, whereas lp_solve is an open source alternative implementing branch-and-bound method for solving integer problems. For performance reasons, lp_solve is only used for debugging purpose whereas CPLEX is applied for evaluation of the mechanism.

5.5 Protocol

A dedicated protocol, named CombiSVMP (stands for Combinatorial Simple Virtual Market Protocol), has been designed to exchange information between the marketplace server and the participant agents. Its basic design is derived from SVMP, which is developed for U-Mart v2 system [44]. CombiSVMP extends SVMP to be capable of (1) simultaneous interaction with multiple auctioneers, (2) combinatorial trading of multiple goods and (3) forward trading of multiple timeslots.

Table 5-1 lists the commands defined by CombiSVMP. Most of the commands have AuctioneerName property as a string to specify the auctioneer with whom the agent communicates. Some non-trivial commands and properties are described below.

- The OrderRequest command is used to place a sell/buy order. It has OrderSpec property, which includes arbitrary number of goods[4] and their

[4] The number of goods is limited to one for sell orders

properties like quantity, earliest timeslot, latest timeslot and total number of timeslot to provide/use the goods.

- The OrderStatus command is used to inquire the outcome of past orders. It returns OutcomeSpec property, which includes two arrays (vectors) representing the quantities and the prices to be sold at each timeslots.
- The MarketPrice command is used to inquire the historical information of the market. It returns a specified number of histories of MarketPriceTable's, which includes the market price (i.e. the average sold price per unit per hour) for each good and each timeslot.

Such an exhaustive capability of CombiSVMP allows participant agents to make full use of combinatorial/forward trading features of the proposed marketplace.

Table 5-1 Specification of CombiSVMP commands

Command	Arguments	Returns
OrderRequest	UserID, AuctioneerName, SellOrBuy, OrderPrice, OrderSpec	Order ID
OrderStatus	UserID, AuctioneerName	OrderID, OrderTime, AuctioneerName, SellOrBuy, OrderPrice, OrderSpec, OutcomePrice, OutcomeSpec
MarketStatus	AuctioneerName	StatusCode
MarketPrice	AuctioneerName, HistorySize	Date, Session, Slot, MarketPriceTables
ServerStatus	UserID	Date, Session, StatusCode
ServerDate	-	Date, Session
ServerTime	-	WallclockTime

Remarks:

OrderSpec = "Good:Quantity:Earliest:Latest:TotalLength;"+

OutcomeSpec = "Good:Quantity:Earliest:Latest:SoldQuantities:SoldPrices;"+

SoldQuantities = "SoldQuantity[Earliest], ..., SoldQuantity[Latest]"

SoldPrices = "SoldPrice[Earliest], ..., SoldPrice[Latest]"

MarketPriceTables = MarketPriceTable+

MarketPriceTable = "Good:Offset:MarketPrice[1], ..., MarketPrice[T];"+

"+" indicates arbitrary repetition.

Chapter 6 Experimental Study

To confirm the advantage of the proposed mechanisms, three experiments have been carried out: (1) validation of combinatorial allocations, (2) estimation of scalability, and (3) evaluation of market performance. This chapter demonstrates these results and discusses the excellence and weakness of the proposed mechanism.

All the simulations in this chapter are carried out using our W-Mart simulator run on the environment shown in Table 6-1.

Table 6-1 Simulation environment

CPU	AMD Opteron 8218 HE (2.6 GHz)
RAM	32GB
OS	CentOS 5.1 (Linux kernel 2.6.18-92.el5)
JRE	Sun Java SE 1.6.0_11
Solver	ILOG CPLEX 11.200

6.1 Validation of Combinatorial Allocation

The proposed mechanism enables combinational allocations for workflows and co-allocations. This section investigates two simple cases to see how combinational allocations are achieved in the forward market and in the spot market.

6.1.1 Settings

The forward market is assumed to have four timeslots (i.e. from zero o'clock to four o'clock). Two kinds of services are offered by three providers: the provider 1 offers service A; the provider 2 and the provider 3 offer service B with different prices. Two users require these services in different manners: the user 1 needs the services A

Table 6-2 Simulation settings for validation of combinatorial allocation

		Forward	**Spot**
Market	Forward delivery hours	$T = 4$ slots	
Seller	Service types	$\{g_{m_1}, g_{m_2}, g_{m_3}\} = \{A, B, B\}$	
	Quantity of a service	$\{q_{m_1}, q_{m_2}, q_{m_3}\} = \{40, 30, 30\}$ units	
	Order price	Shown in Figure 6.1 thru Figure 6.4	
Buyer	Service types	$G \in \{$A, B$\}$	
	Number of task in a workflow	$H = 2$	
	Quantity of service required by a task	$q_j = [10, 30]$ units	
	Order price	Shown in the figures	
	Length of a workflow	$l_j = [3, 4]$ slots $l_j = d_j - b_j + 1$	$l_j = 1$ slot
	Length of a task	Shown in Figure 6.1 thru Figure 6.4	

and B simultaneously for co-allocation; the user 2 needs the services A and B sequentially for a workflow. The quantities, valuations and start/finish times of each service are shown in Figure 6.1 and Figure 6.3. The required runtime of the task equals (finish time -start time), which means that no interruption occurs. Table 6-2 shows the formulations of the simulation parameters.

Only one timeslot is available for the spot market. The providers and the services are the same as those of the forward market. The user 1 and the user 2 require the same combination of the services A and B, but the valuation of the user 1 is higher than that of the user 2.

6.1.2 Results

Figure 6.1 and Figure 6.2 respectively show orders and the allocation results in the forward market. These results show that orders from all the users are fulfilled. In particular, the order from the user 2 consists of two tasks in a workflow – a task of the service A and one of the service B – and the services are properly allocated to the tasks. These results indicate that the proposed mechanism using the combinational auction properly allocated the services to workflow tasks. Note that the provider 3 won the competition to sell the service B for the user 1 in timeslot 2 because he priced it lower than the provider 2 did and therefore generated more total welfare.

Figure 6.3 and Figure 6.4 respectively show orders and the allocation results in the spot market. The supply of the service B is less than the demand. As a result, the user 2 lost the competition and bought nothing. Indeed the provider 1 still has enough capacity for the service A, but it is not allocated to the user 2 since it does not fulfill the combinational order of the user 2

6.1.3 Summary

It is confirmed through this preliminary experiment that the proposed

mechanism properly allocates the services amongst multiple providers, multiple users, multiple kinds of services and combinatorial requirements for them. The outcome of the mechanism is reliable. But how long time does it take? The next section will evaluate the overhead to deal with a large number of participants.

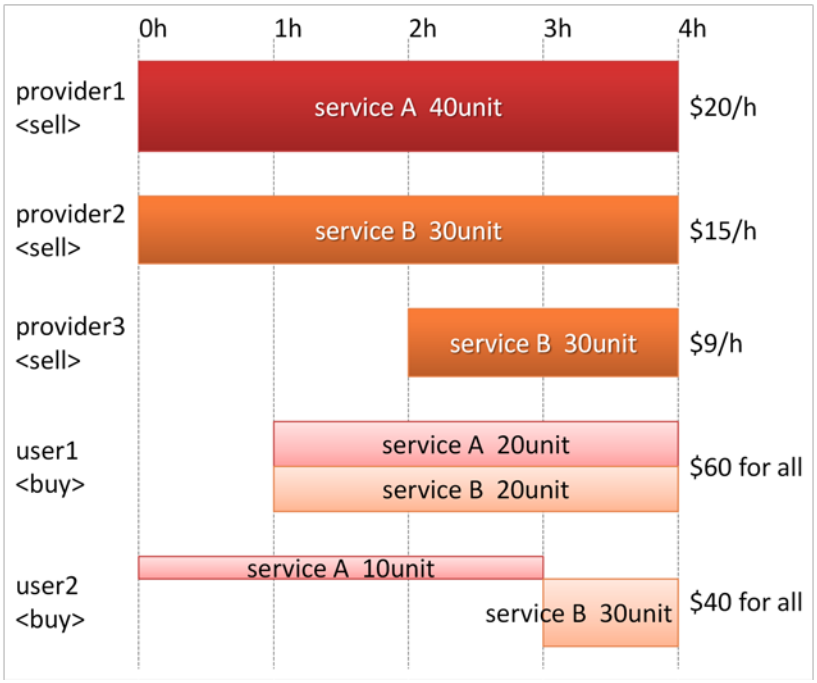


Figure 6.1 Forward orders

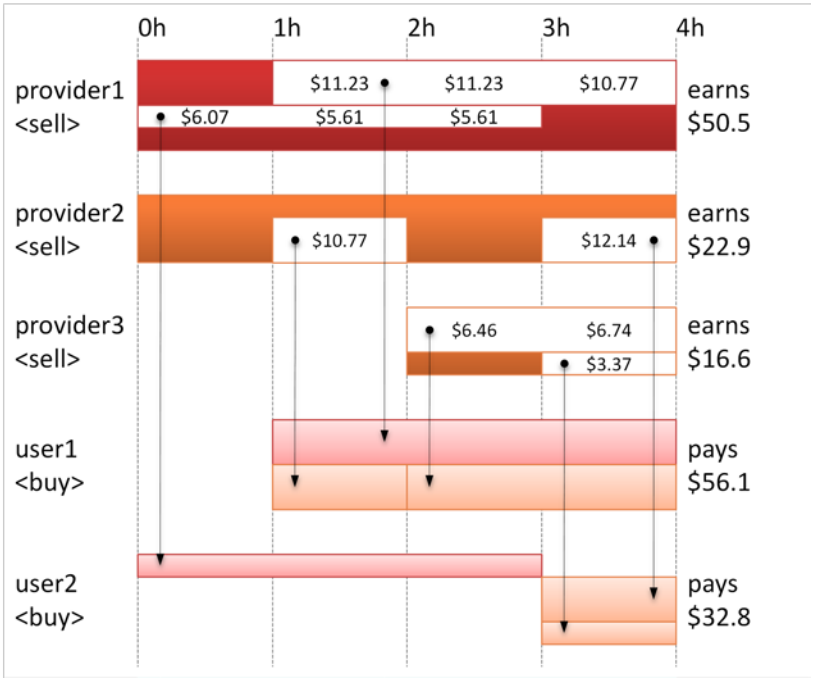


Figure 6.2 Forward allocation

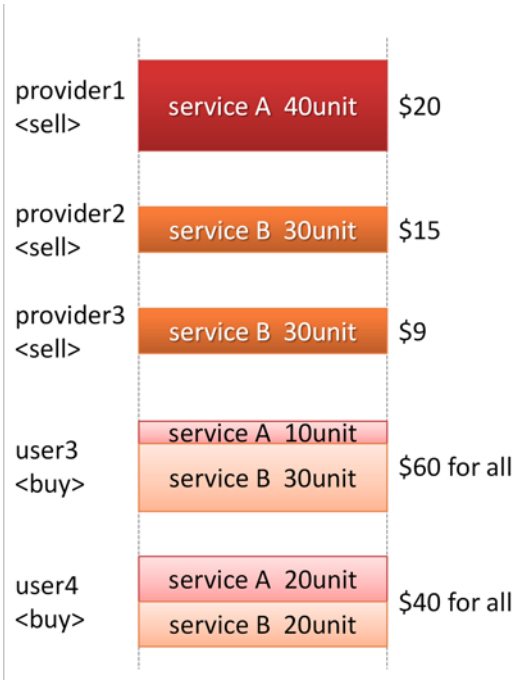


Figure 6.3 Spot orders

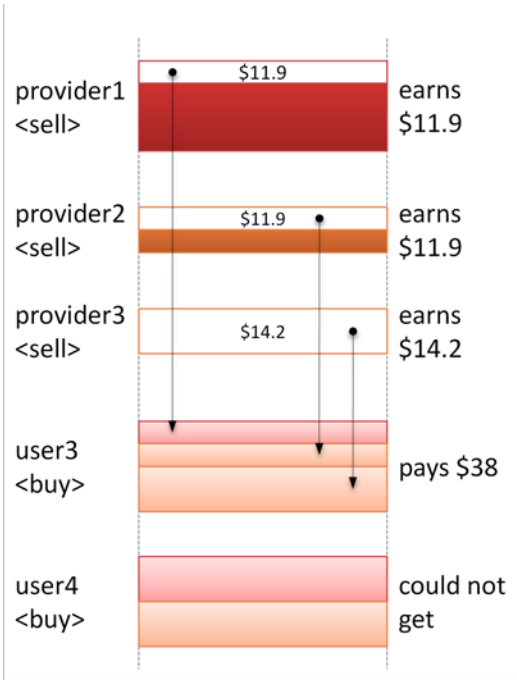


Figure 6.4 Spot allocation

6.2 Estimation of Mechanism Overhead

Mixed integer programming tends to consume a long time when faced with a large problem. This section evaluates the overhead of the proposed mechanism in order to confirm its practicality in a cloud computing environment. The evaluation assesses the impact of the number of users and timeslots on the runtime.

6.2.1 Settings

The simulation is carried out by generating a set of orders and running the market mechanism. Since the evaluation aims to assess the overhead, the rounds are assumed to be independent; i.e., the result of matchmaking of orders does not affect the next orders. The number of timeslots has a range of $\{1,24,120,240,480,720\}$. The case of $\#\text{slots} = 1$ represents trading in the spot market, and other cases represent trading in the forward market. The actual time span covered by timeslots depends on the length of the timeslot. For example, $\#\text{slots} = 720$ represents one month with a timeslot of one hour, or represents one year with a timeslot of 12 hours. The Japanese electricity exchange is referred as an example for the time granularity. This extent of granularity should be applicable to the cloud computing environment.

The number of providers is set to 10, while the number of users has a range of $\{100, 400, 700, 1000\}$. Figure 6.5 shows the sell orders of the providers. Each provider offers a unique service and all the services are available anytime. Figure 6.6 shows the buy orders of the users. Each user requires one to five services chosen randomly out of 10 services to be co-allocated. The task length varies from one to 12 timeslots. The time margin between ordering and starting a task varies from zero to (#slots-12) timeslots. This setting is intended to reflect the current situation of cloud computing, where some big companies provide their own services and many small

consumers use services to execute their tasks.

Other parameters are set constant for the sake of simplicity. The quantity of a service is 100 units for selling and one unit for buying. The valuation of a service is $\$1/(\text{slot}\cdot\text{unit})$ for selling and $\$3/(\text{slot}\cdot\text{unit})$ for buying. This setting means a loose supply-demand condition with no price competition, where the buyer's requirements are likely to be fulfilled. The simulation was conducted 10 times for each setting with different random seeds, and the average results are presented.

Table 6-3 Simulation settings for estimation of mechanism overhead

		Lower load	**Higher load**
Market	Forward delivery hours	$T \in \{1,24,120,240,480,720\}$ slots ($T = 1$ means a spot trading)	
Seller	Service types	$G \in$ {A, B, C, D, E, F, G, H, I, J}	
	Quantity of a service	$q_{i,k} = 100$ units	
	Order price (per unit per hour)	$\frac{v_i}{q_{i,k}(d_{i,k}-b_{i,k}+1)} = 1$ dollar	
Buyer	Service types	$G \in$ {A, B, C, D, E, F, G, H, I, J}	
	Number of task in a workflow	$H \in \{1,2,3,4,5\}$	
	Quantity of service required by a task	$q_{j,k} = 1$ unit	$q_{j,k} = [1,100]$ units, uniform dist.
	Order price (per unit per hour)	$\frac{v_j}{q_{j,k}l_{j,k}} = 3$ dollars	
	Length of a workflow	$l_{j,k} = [1,12]$ slots, uniform dist. $l_{j,k} = d_{j,k} - b_{j,k} + 1$	
	Length of a task	$l_{j,1} = \cdots = l_{j,H} = l_j$ hours	
	Margin	$[0, T-12]$ slots, uniform dist.	

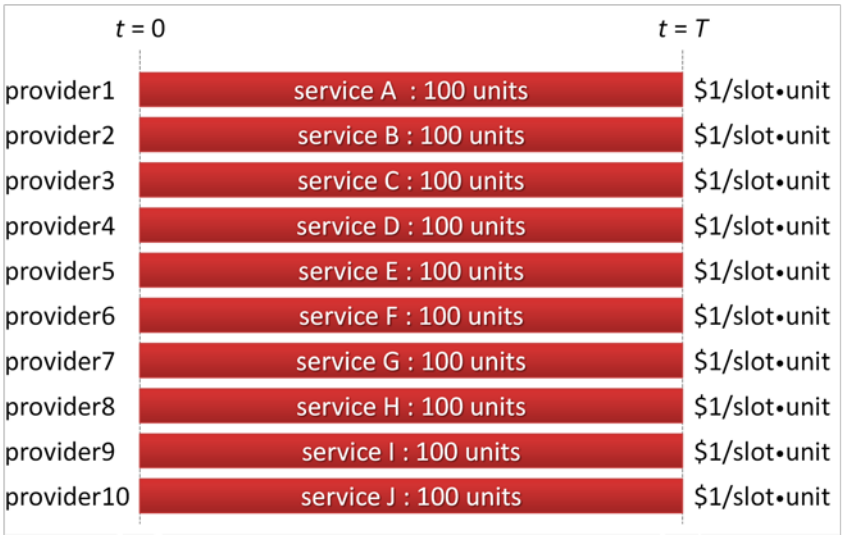


Figure 6.5 Sell orders

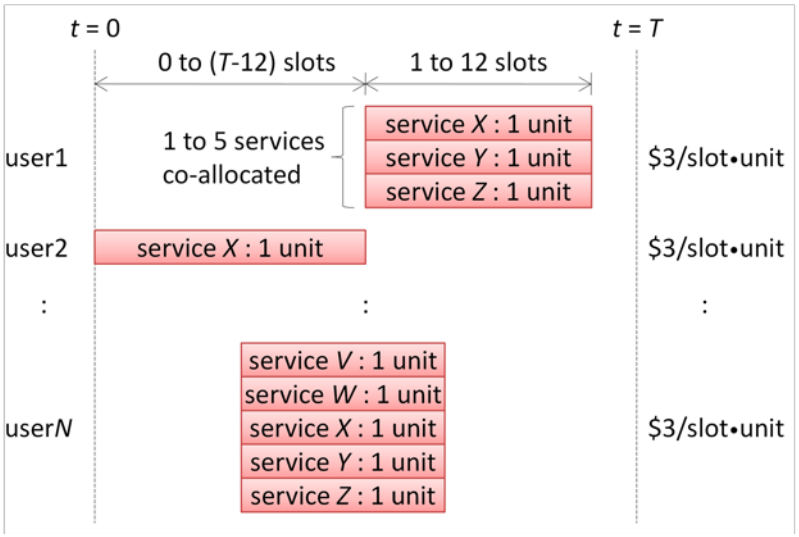


Figure 6.6 Buy orders

Table 6-3 shows the formulation of the simulation parameters.

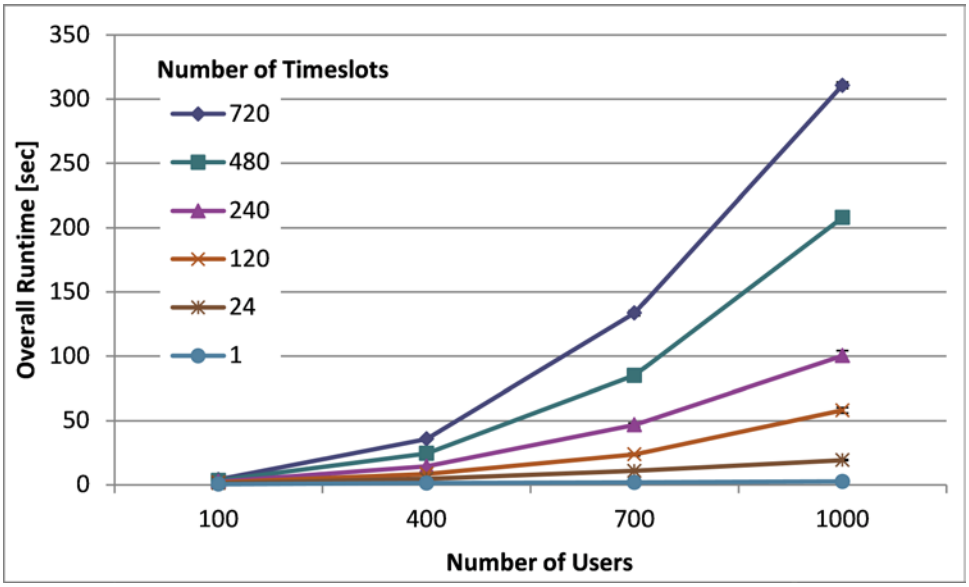


Figure 6.7 Overall runtime (lower load)

6.2.2 Results

A. Lower Load

First a basic evaluation is carried out as described above, which simulates relatively easy supply-demand conditions.

- Overall Runtime

Figure 6.7 shows the overall runtime consumed by the market mechanism to perform a round of matchmaking. For the forward market, it takes more than five minutes with 720 timeslots and 1000 users. However, it is still shorter than the length of a timeslot, which is assumed to be one hour or 12 hours. The result for the spot market is shown as $\#\text{Timeslots} = 1$. For the spot market, it takes less than one second. The overall runtime is essentially proportional to $|M| \times |N| \times |G| \times T$, which is the number of iterations to build the model and parse the results.

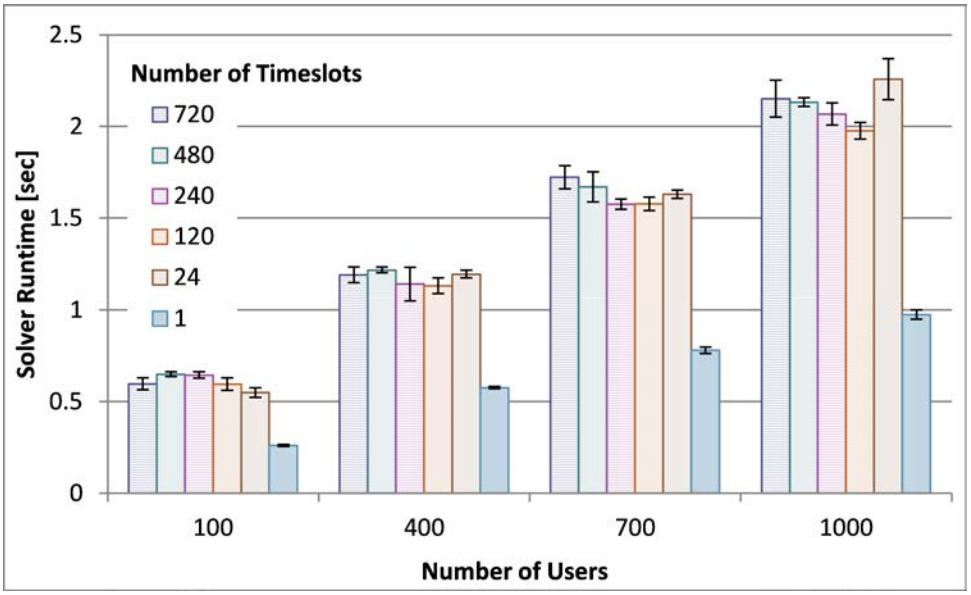


Figure 6.8 Solver runtime (lower load)

- Solver Runtime

Figure 6.8 shows the runtime of the solver, i.e. excluding the time to build the model, etc. It takes less than 3 seconds in the worst case. This means the translated problems are so easy for MIP solver. The solver runtime is mainly affected by the difficulty to find the optimal solution, which is more sensitive to the number of conflicted orders than the number of timeslots. Further investigation is needed about the intensity of confliction, which is shown in the next chart.

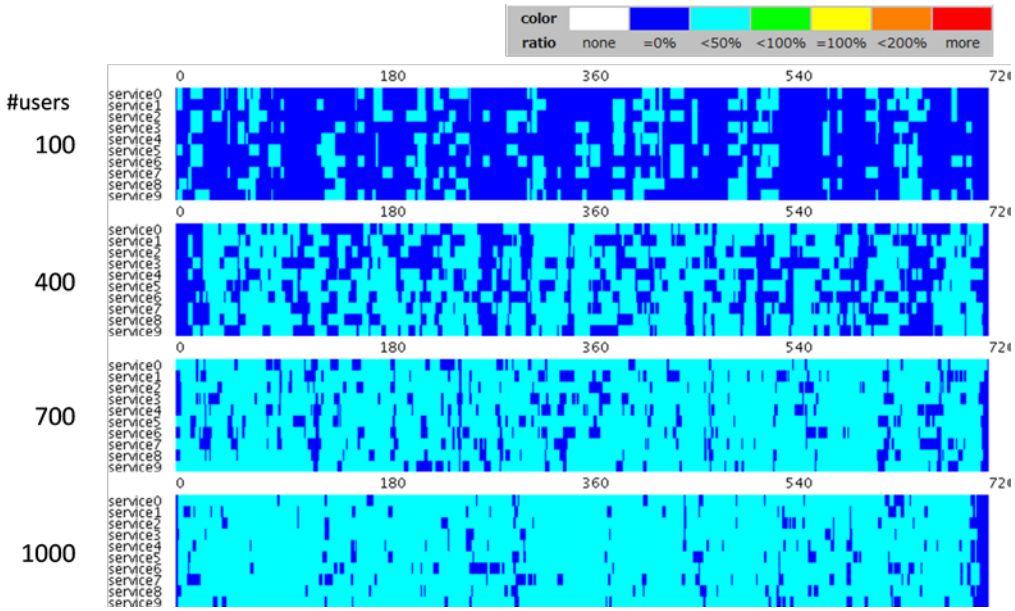


Figure 6.9 Example demand/supply ratio (lower load)

- Demand/supply Ratio

Figure 6.9 depicts the demand/supply ratio (D/S) of an arbitrary simulation run of the lower-load experiments. The X axis indicates the timeslot (720 in total) and Y axis indicates the type of service, and the color block indicates the D/S. We can see the D/S is less than 50% throughout the simulation, that is, almost no competition occurs during the trading. Therefore, we can consider that the results of lower-load experiments show mainly the overhead of the wrapping W-Mart mechanism, which converts the orders into the MIP and vise versa, rather than the performance of underlying MIP solver.

It is needed to evaluate the capacity of MIP solver when faced to larger problem derived from higher demand/supply ratio. The next section will investigate it.

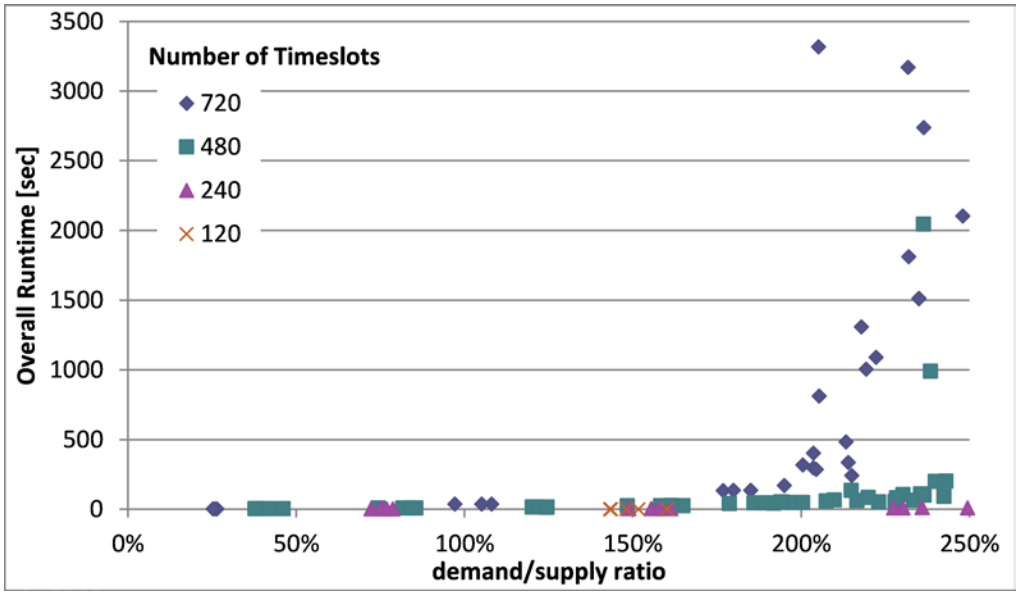


Figure 6.10 Overall Runtime (higher load)

B. *Higher Load*

The quantity of buying orders is changed from 1 unit to a random value chosen uniformly within $[1..100]$ units (on average 50.5 units), whereas the quantity of selling orders are 100 units. This makes the demand-supply conditions very hard and competitions among users are expected to occur frequently. Other parameters are identical to those of the lower-load experiment.

- *Overall Runtime*

Figure 6.10 shows the overall runtime of the market mechanism to perform a round of matchmaking. One plot indicates one simulation run. The x axis indicates the demand/supply ratio. When the D/S is below 200%, all the simulation runs finishes within 60 seconds. When the D/S is above 200%, in contrast, some of the simulation eventually begins to consume a very long time.

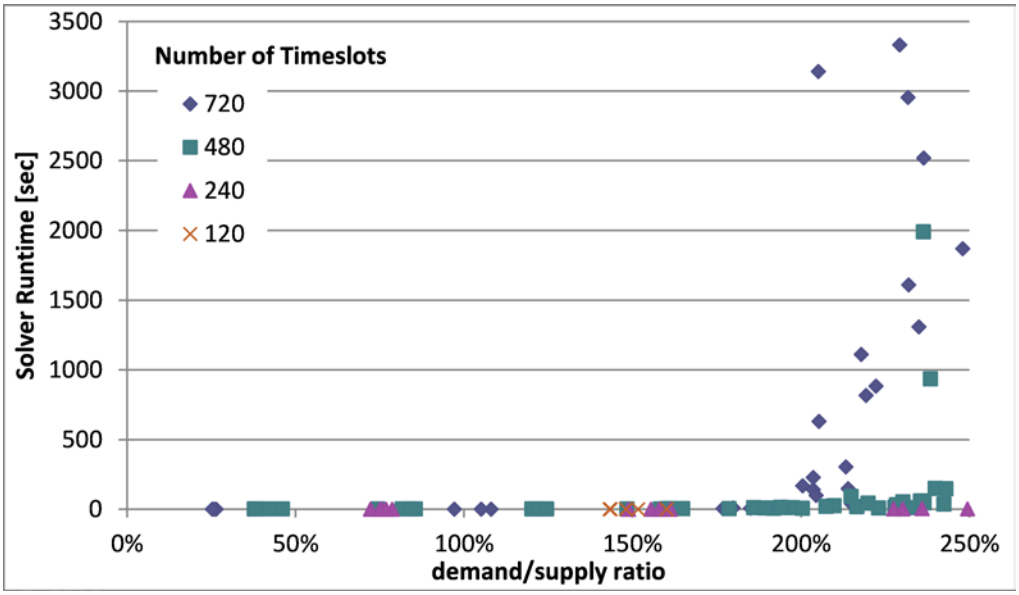


Figure 6.11 Solver runtime (higher load)

- Solver Runtime

Figure 6.11 shows the runtime of the solver with the same scenario. It also jumps up when the D/S exceeds 200%. Some of them is over 3600 seconds and is not plotted on the chart[5]. The largest overhead of W-Mart mechanism (i.e. the difference between the overall runtime and the solver runtime) is 235 seconds at $D/S = 2.48$, when the solver spends 1870 seconds with 10 providers and 900 users. We can consider that the runtime of the solver itself becomes dominant when the solver encounters a difficult situation. Note that 1870 seconds is still shorter than the length of a timeslot in our trading schedule (see Figure 4.3).

[5] The solver's runtime is limited to 3600 seconds in this experiment.

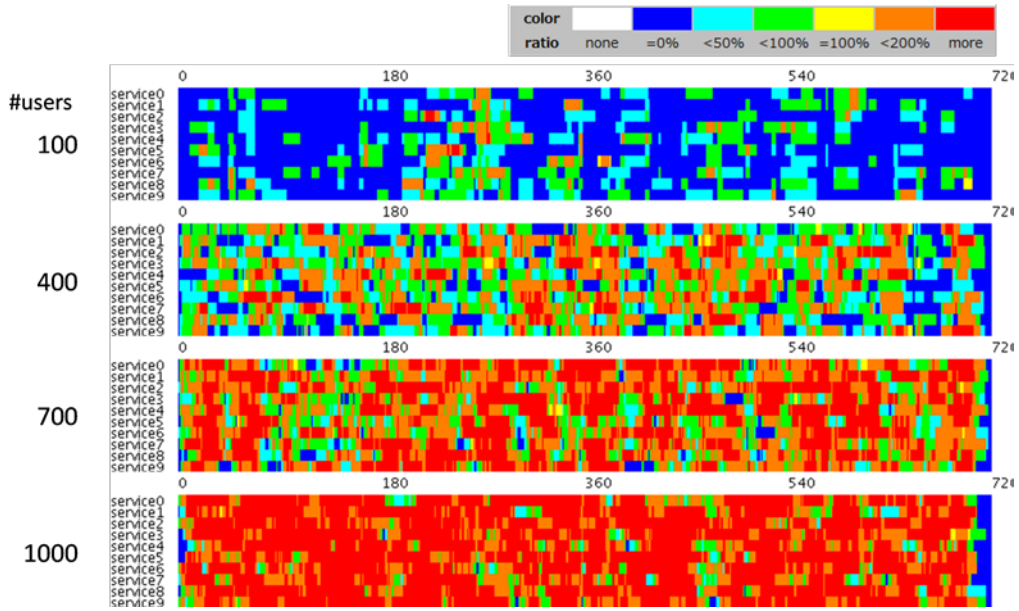


Figure 6.12 Example demand/supply ratio (higher load)

- Demand/Supply Ratio

Figure 6.12 depicts the demand/supply ratio (D/S) of an arbitrary simulation run of the higher-load experiments. The D/S grows up as the number of users increase, and the higher D/S leads to the more runtime of the solver. Note that the runtime of the solver can be exponential to the number of collisions for resources, which is not necessarily proportional to the D/S ratio.

6.2.3 Summary

The simulation results show that the proposed mechanism will scale up to 720 timeslots, 1000 users, 10 providers and 10 services. With 200% or more excess load conditions, however, the MIP solver tends to consume a much longer time: typically 1 hour or longer at 250% load with 720 timeslots. Nevertheless, remember that the proposed marketplace mechanism secures enough time to perform matchmaking: 1 hour for spot and 22 hours for forward trading.

6.3 Evaluation of Market Performance

Three experiments are contracted to see the performance of the proposed mechanism. The first experiment runs a single market (the forward market or the spot market) and evaluates how the combinatorial auction improves the performance. The second and the third experiments run the forward market and the spot market simultaneously to see the interaction between them and to evaluate the performance of the whole marketplace with various users.

Figure 6.13 illustrates the difference between these scenarios. First, in the

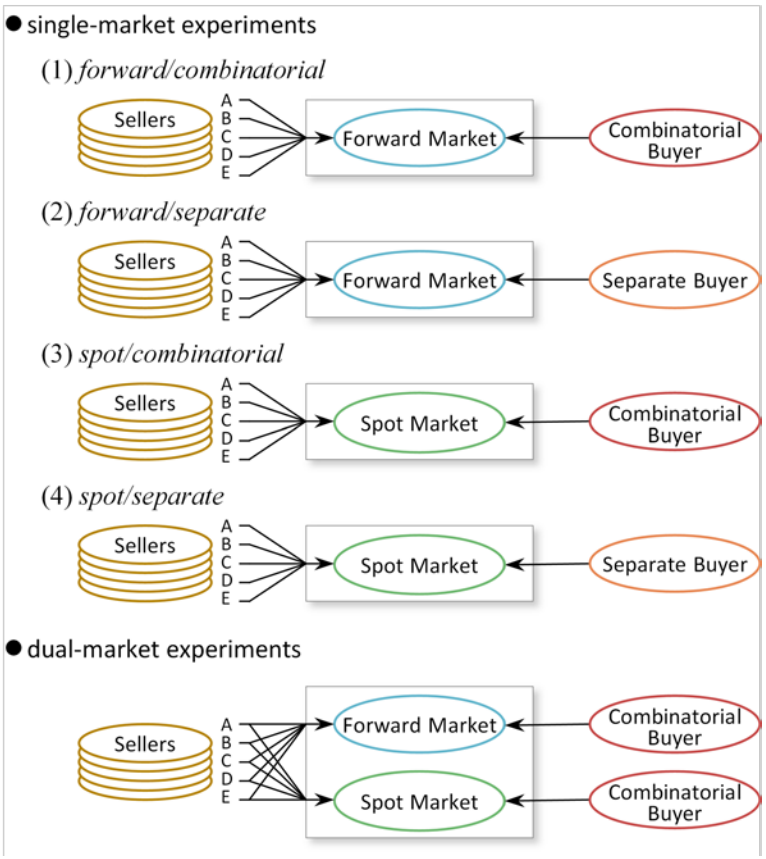


Figure 6.13 Overview of the experiments

single-market experiment, the effectiveness of the combinational auction is studied by running four scenarios: (1) forward/combinatorial, (2) forward/separate, (3) spot/combinatorial and (4) spot/separate. Each user places a combination of orders for services in the scenario (1) and (3). The forward market and the spot market are used in the marketplace in the scenario (1) and (3), respectively. In the scenarios (2) and (4), each user places orders for services separately.

Second, in the dual-market basic experiment, the practicality of running two markets simultaneously is studied, where the forward contracts should have a priority over the spot contracts. Combinatorial orders are enabled in this scenario.

Third, in the dual-market extensive experiment, the characteristics of the marketplace running two markets is studied.

6.3.1 Settings

The simulations are carried out with the settings described below. Table 6-4 summarizes all the parameters used in the simulations.

A. Market

One timeslot is one hour and one day is 24 timeslots. The market operates for 30 days (720 hours). The forward market deals with seven days (168 hours) of future services and clears a round at every midnight. The spot market deals with one hour of services and clears a round every hour. This setting exactly implements the trading schedule described in Section 4.

Table 6-4 Simulation settings for evaluation of market performance

			Single market	Dual market
Market	Duration of operation		$T = 720$ hours	
	Forward delivery days		$F = 7$ days	
Seller	Service type		$G \in$ {A, B, C, D, E}	
	Quantity of a service		$q_i = 100$ units	
	Order price (per unit per hour)		$\frac{v_i}{q_i(d_i - b_i + 1)} = 1$ dollar	
Buyer	Service type		$G \in$ {A, B, C, D, E}	
	Number of task in a workflow		$H \in \{2, 3, 4, 5\}$	
	Quantity of service required by a task		$q_j = [1, 100]$ units, uniform dist.	
	Order price (per unit per hour)		$\frac{v_j}{q_j l_j} = [2, 10]$ dollars, uniform dist.	
	Length of a workflow		$l_j = [2, 24]$ hours, exponential dist. ($\lambda = 0.25$) $l_j = d_j - b_j + 1$	
	Length of a task	Leading tasks	$l_{j,1} = \cdots = l_{j,H-1} = [1, l_j - 1]$ hours, uniform dist.	
		Following task	$l_{j,H} = l_j - l_{j,1}$ hours	
	Margin	Forward	$[2, 7]$ days, uniform dist.	
		Spot	2 hours	
	Arrival rate (Poisson Arrival)	Forward	$\lambda \in \{1, 3, 5, 7, 10, 14, 20, 30, 40, 50\}$	$\lambda = 10$
		Spot		$\lambda \in \{1, 3, 5, 7, 10, 14, 20, 30\}$

B. Seller Agents

There are five sellers with different kind of services, namely service A, B, C, D, and E. A seller has an ability to provide constant units of his service every hour. The order price is fixed to $\$1$ per unit per hour, for the sake of simplicity. A seller agent attempts to sell all amount of his service as early as possible. For instance, after he ordered 100 units and contracted 30 units to sell in one round, he will order to sell remaining 70 units in the next round.

C. Buyer Agents

Each buyer has their own applications presented by workflows. Each workflow consists of two phases of tasks, namely the leading task(s) and the following task, as shown in Figure 6.14. Each task requires service A, B, C, D or E, exclusively. The overall length of a workflow follows the exponential distribution with $\lambda = 0.25$, where the minimum length is two hours and the maximum is 24 hours. This means that 50% of the workflows have four hours or shorter length. The lengths of the leading tasks are set randomly within the overall length of the workflow and the following task spends the remainder. The valuation of a task follows the uniform distribution between 2 and 10 cents per unit per hour for each task. The order price is the sum of the valuations of all tasks in the workflow.

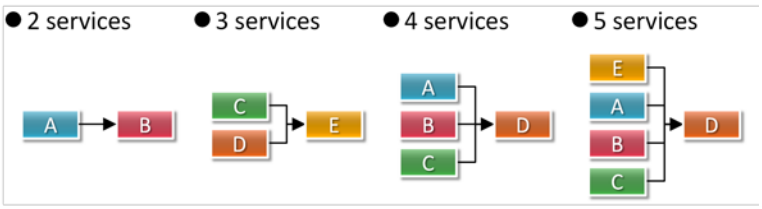


Figure 6.14 Shapes of workflow

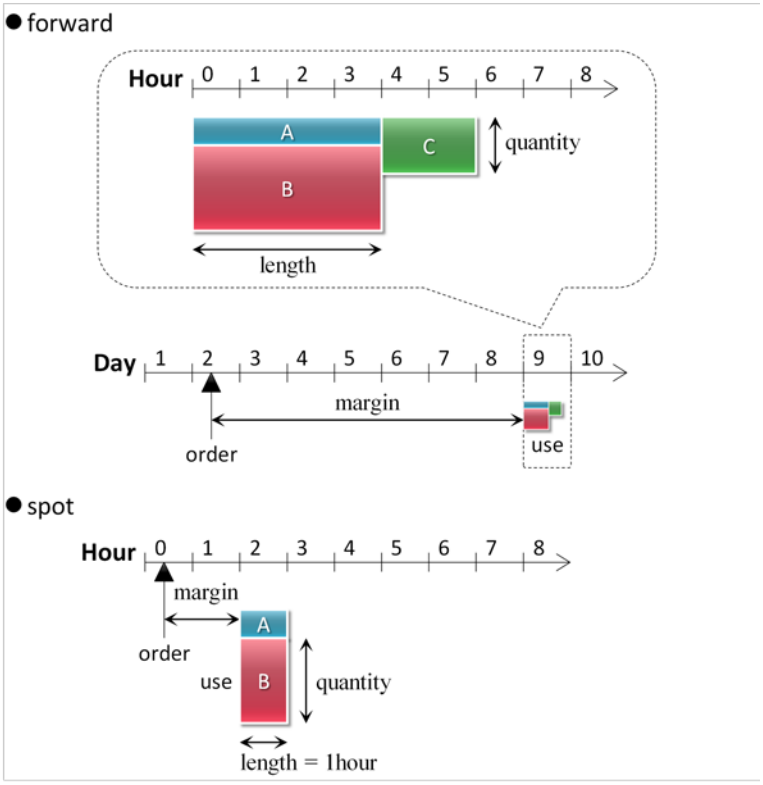


Figure 6.15 Buyer order

The user places an order for services before the time the user actually start to use them. The margin of time between ordering and starting the services follows the uniform distribution between two and seven days for the forward orders, and is fixed to two hours for the spot orders. The quantity of each service in an order follows the uniform distribution between 1 and 100 units. Figure 6.15 shows examples of orders to the forward market and to the spot market. Arrival of workflows is assumed to follow the Poisson Arrival with parameters shown in Table 6-4, where λ denotes the expected number of arrival in one day and k denotes the maximum number of occurrences in one day. The simulation run is conducted 100 times for each arrival rate and the average results are shown.

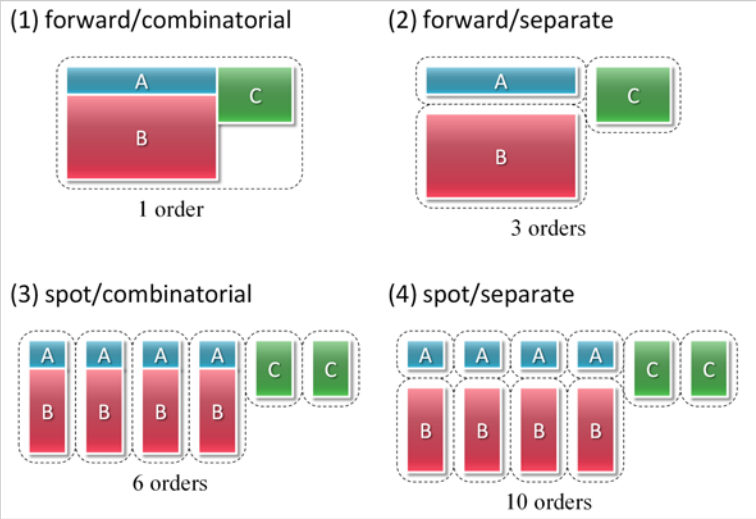


Figure 6.16 Order division

A buyer agent attempts to buy all services in a workflow. In forward/combinatorial scenario he orders all services at once as a bundle; in other scenarios he divides the bundle into a set of orders in an appropriate manner. Figure 6.16 illustrates how to divide a workflow depending on the market mechanism. Here, a user demands a workflow of three services, for example, starting with four hours of A and B followed by two hours of C. In (1) the forward/combinatorial scenario he puts one order for all services on the forward market at once. In (2) the forward/separate scenario he puts three orders for each service on the forward market at the same time, i.e. each order is processed independently. In (3) the spot/combinatorial scenario he puts six orders for each timeslot on the spot market for six times. In (4) the spot/separate scenario he puts 10 orders for each service and timeslot on the spot market for six times. The workflow is fulfilled if he succeeds to reserve all services in the workflow; otherwise the workflow is not fulfilled and reserved services are wasted, i.e. paid but not utilized. In other words, the wasted services give the buyer no benefit while consuming his budget.

6.3.2 Performance Metrics

The following metrics are used to evaluate the performance of the market mechanisms.

- *Demand/supply ratio (D/S)*

Demand/supply ratio (D/S) indicates a load on the overall system, in other words, the total quantity (i.e. the product of quantity and total runtime) of services requested by users compared to those offered by providers. It also aggregates all kinds of service. The aggregation of different services is acceptable since they have an identical setting in our experiments. D/S is computed by (19).

$$D/S = \frac{\sum_{j \in N} \sum_{k \in G} q_{j,k} l_{j,k}}{\sum_{i \in M} \sum_{k \in G} q_{i,k} \left(d_{i,k} - b_{i,k} + 1 \right)} \tag{19}$$

- *Workflow completion rate (WC)*

Workflow completion rate (WC) indicates the rate of the number of workflows fulfill their requirements compared to the total number of workflows. A higher rate means better performance. Note that WC compares the number of workflows, not the quantity of services in workflows. WC is computed by (20).

$$WC = \frac{\sum_{j \in N} u_j}{|N|} \tag{20}$$

- Cost performance (CP)

Cost performance (CP) indicates the users' total valuation fulfilled (excluding wasted services) compared to the total payments (including wasted services) by the users. A higher value means better performance. CP is computed by (21).

$$CP = \frac{\sum_{j \in N} v_j u_j}{\sum_{j \in N} p_j} \tag{21}$$

- Global utilization (GU)

Global utilization (GU) indicates the total quantity of services utilized in the market (i.e. reserved and not wasted) compared to those offered by providers. A higher utilization means better performance. GU is computed by (22).

$$GU = \frac{\sum_{j \in N} \sum_{k \in G} q_{j,k} u_j}{\sum_{i \in M} \sum_{k \in G} q_{i,k}} \tag{22}$$

- Market price (MP)

It indicates an average price per unit per hour decided in the market. The users/providers actually pays/earns this amount of money. A lower price means better for users. MP is computed by (23).

$$MP = \frac{\sum_{i \in M} p_i}{\sum_{i \in M} \sum_{j \in N} \sum_{k \in G} \sum_{t \in T} y_{i,j,k,t} q_{i,k}} \tag{23}$$

Note that the minimum MP in the experiments is 3.5. The reason is that an order price is $\$1$ for a selling order and $[\$2 \dots \$10]$ in uniform distribution ($\$6$ on average) for a buying order; thus, the K-pricing scheme calculates $(1 + 6)/2 = 3.5$.

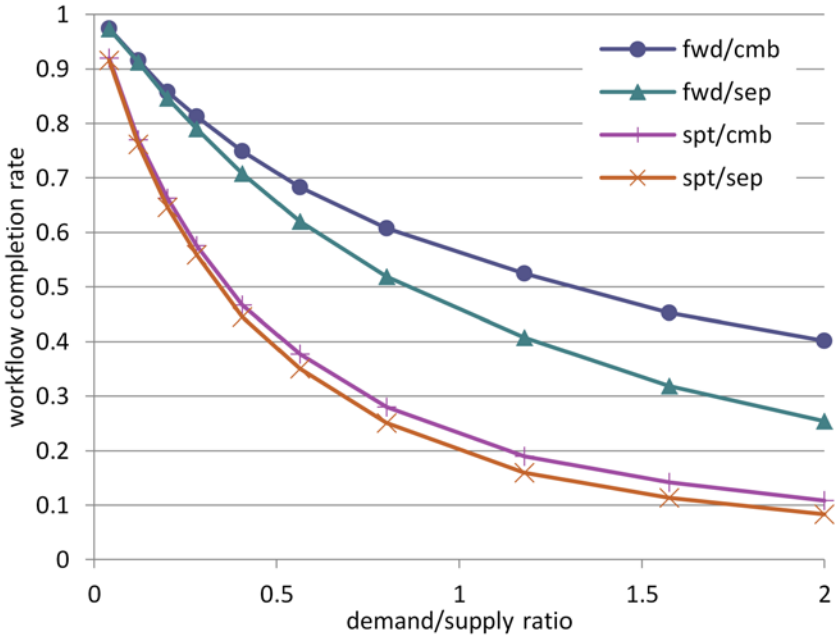


Figure 6.17 Workflow completion rate (single market)

6.3.3 Results

A. Single Market

First the results of the single-market experiments are shown to compare performance of four market mechanisms: forward/combinatorial (fwd/cmb), forward/separate (fwd/sep), spot/combinatorial (spt/cmb), and spot/separate (spt/sep).

- Workflow Completion Rate

Figure 6.17 shows the workflow completion rate (WC). First we can see the advantage of the forward market mechanism. Users have more opportunity to buy services that satisfy the users' requests in the forward market. Second, the combinatorial market mechanism shows more advantage compared to the separate market mechanism. In the cloud computing model discussed in this thesis, users

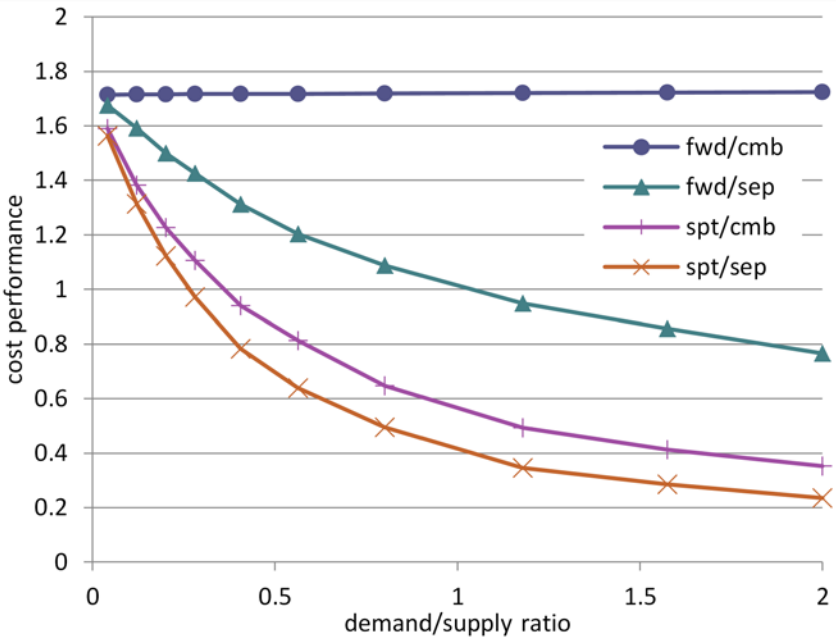


Figure 6.18 Cost performance (single market)

need to reserve services for all tasks in the users' workflow. The combinatorial market releases all services for tasks in the workflow if it fails to fulfill the requirements, while the separate market keeps them. Thus, the separate market significantly wastes services and degrades the performance. It is clear that *fwd/cmb* is the best mechanism to improve WC.

- Cost Performance

Figure 6.18 shows the cost performance (CP). CP indicates the effectiveness of the market mechanism from the users' point of view. The result shows that CP keeps the ideal value in *fwd/cmb* mechanism, because it guarantees the users to profit from all the services they pay for. In other mechanisms, in contrast, CP decreases monotonically because they cannot guarantee the users to complete their workflows while keeping the payments for the fragmented services.

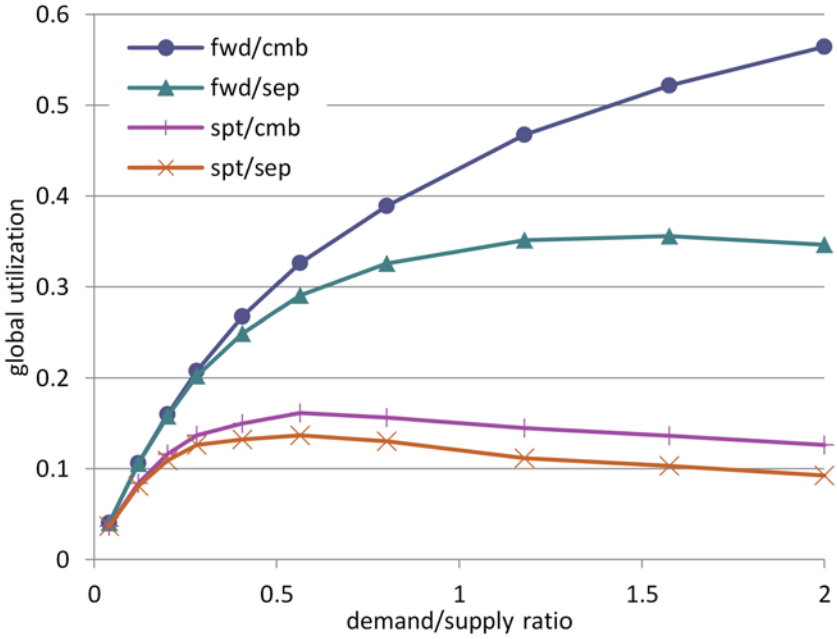


Figure 6.19 Global utilization (single market)

- Global Utilization

Figure 6.19 shows the global utilization (GU). GU indicates the effectiveness of the market mechanism from the providers' point of view. Ideally speaking, GU can be equal to D/S where $D/S \leq 1$ and can be 1 where $D/S > 1$. The result shows that GU increases monotonically in *fwd/cmb* mechanism since it does not waste any services. In *fwd/sep* mechanism the similar trend is observed where $D/S \leq 1$. In the spot market mechanism, in contrast, GU is saturated quickly and begins to decrease because the excessive collision in the spot market makes most of the workflow incomplete and wastes significant amount of services.

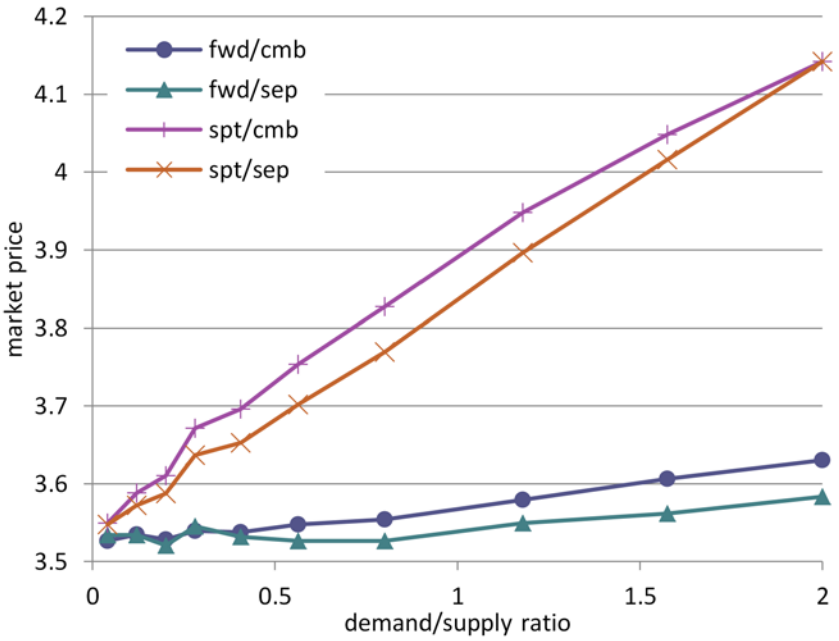


Figure 6.20 Market price (single market)

- Market Price

Figure 6.20 shows the market price (MP). Again we see the advantage of the forward market mechanism, in which the users can buy services at significantly low prices compared to those of the spot market mechanism. Remember that the market mechanism determines winners among users' requests according to their valuation. Users undergo such competitions for every timeslots in the spot market mechanism, buying services at unreasonably high prices and wasting most of them. In the forward market mechanism, in contrast, users experience fewer competitions for entire reservations, resulting in reasonable prices and efficient utilization of services.

B. Dual Market – Basic

Next the results of the dual-market basic experiment are shown to verify the independency between the forward market and the spot market running simultaneously. In this scenario the forward contracts should have a priority over the spot contracts; otherwise the forward trading cannot serve as an advance reservation. It is verified by deploying two users: one takes part in *fwd/cmb* with a constant load ($\mathrm{D}/\mathrm{S} = 0.2$) and the other takes part in *spt/cmb* while changing its load. The providers place the order first at the forward market to sell as much as possible and next at the spot market to sell the remainder.

Figure 6.21 through Figure 6.24 show the workflow completion rate (WC), the cost performance (CP), the global utilization (GU), and the average market price (MP), respectively. Their x axes indicate the aggregate load of two markets. The plots indicate that the performance of *fwd/cmb* does not fluctuate and is not affected by the spot market. Thus the users can rely on the forward market to make an advance reservation. At the same time, the spot market also works well; it contributes to the providers utilizing the remainder resources and to the users procuring immediate resources.

The simulation results with various loads in *fwd/cmb* are shown in the Appendix.

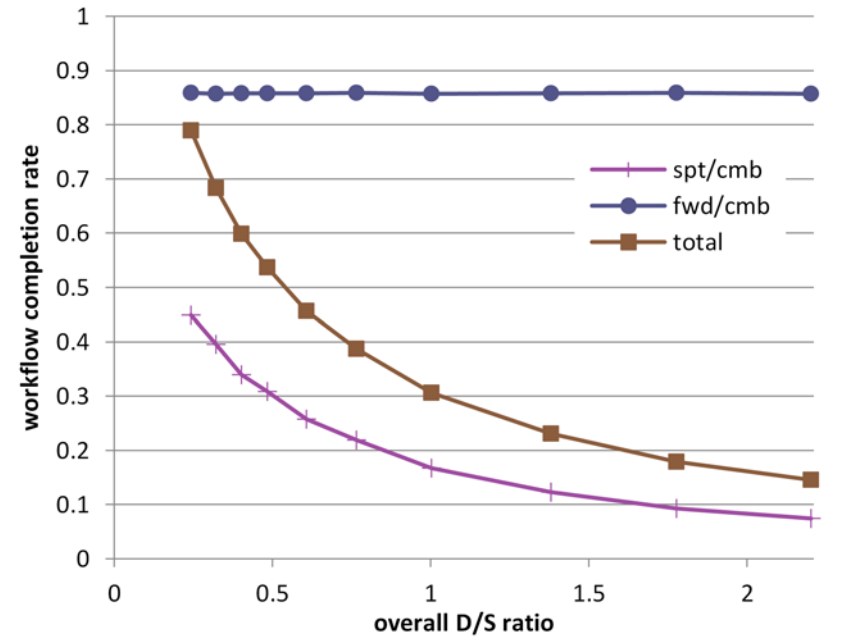


Figure 6.21 Workflow completion rate (dual market basic)

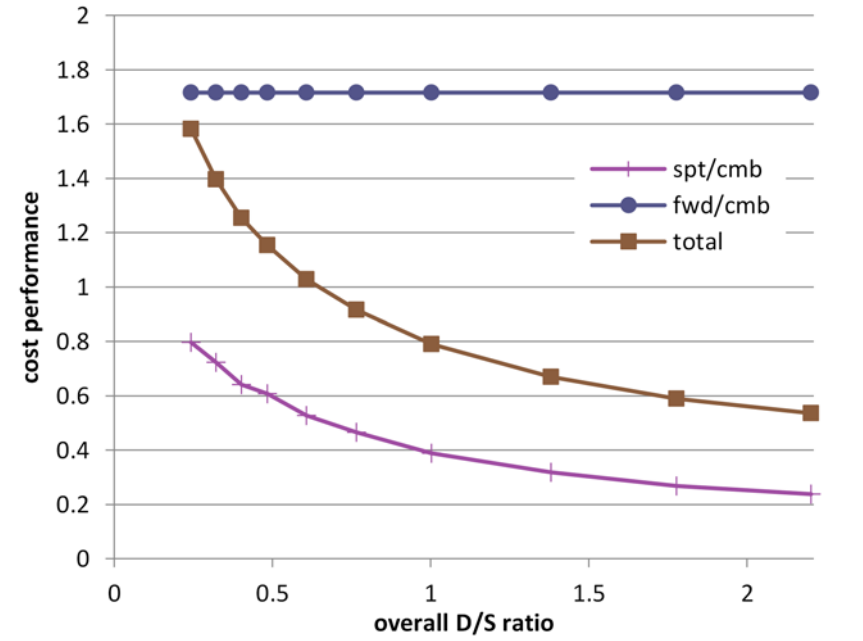


Figure 6.22 Cost performance (dual market basic)

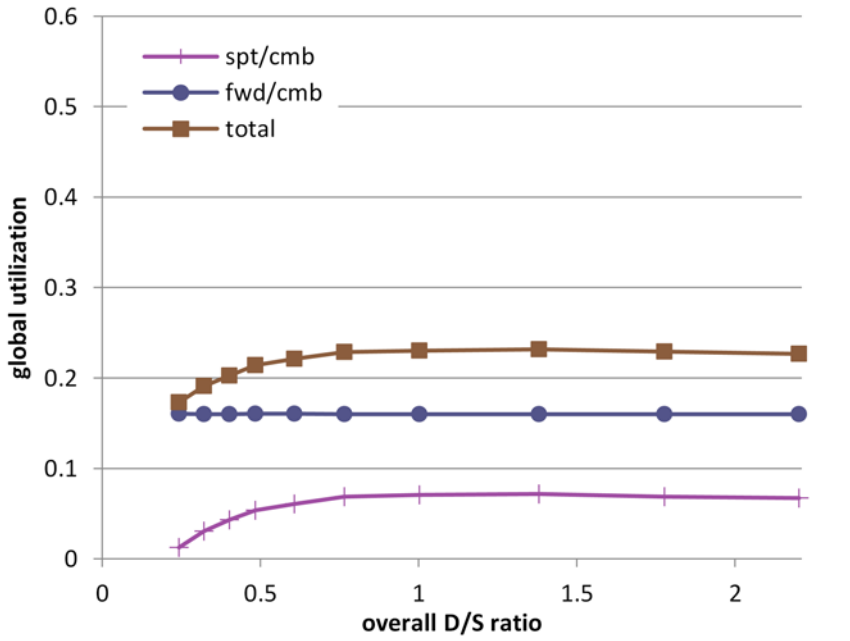


Figure 6.23 Global utilization (dual market basic)

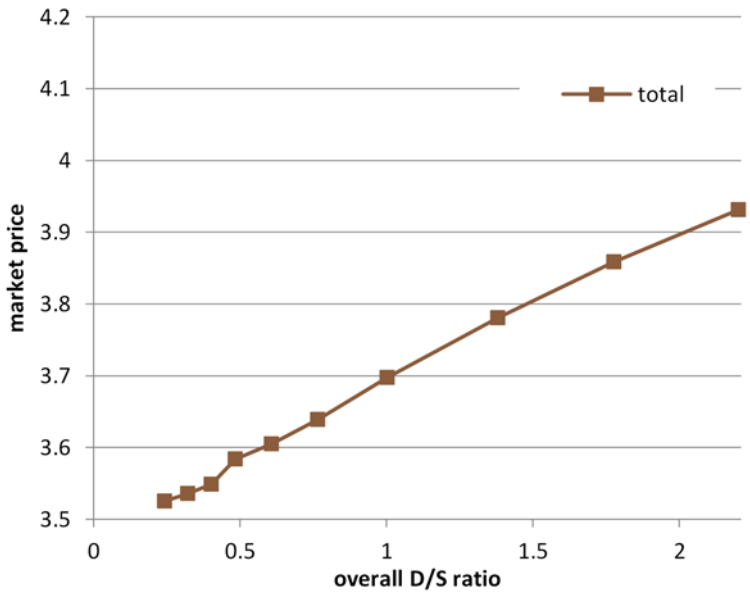


Figure 6.24 Average market price (dual market basic)

C. Dual Market – Extensive

At last the dual-market extensive experiments are carried out to investigate the behavior of the whole marketplace when the users arbitrarily choose the forward market or the spot market to acquire their services. In this scenario a specific percentage of buying orders are put in the forward market and the rest are put in the spot market. By changing the share of the forward/spot markets, a wide range of possible situations are simulated and the characteristics of the whole marketplace are evaluated.

Figure 6.25 through Figure 6.28 show the workflow completion rate (WC), the cost performance (CP), the global utilization (GU), and the average market price (MP), respectively. Their x axes indicate the percentage of the orders put in the forward market, whereas the rest of the orders go to the spot market (i.e. "20% forward orders" means "80% spot orders"). When the load is low enough (i.e. $D/S \leq 0.3$), the spot market and the forward market have no difference in terms of their performance. Both the users and the providers have no reason to use the forward market under these conditions. In contrast, when the load get higher (i.e. $D/S \geq 0.5$), the advantage of the forward market becomes clear; WC and CP increases monotonically to the share of the forward market. MP drops sharply by shifting a half or less percentage of orders from the spot market to the forward market (i.e. $0\% \rightarrow 40\%$ forward orders). GU clearly depicts the disadvantage of the spot market against high load (i.e. $D/S > 0.5$); it rather decreases to the D/S ratio when 40% or less orders go to the forward market.

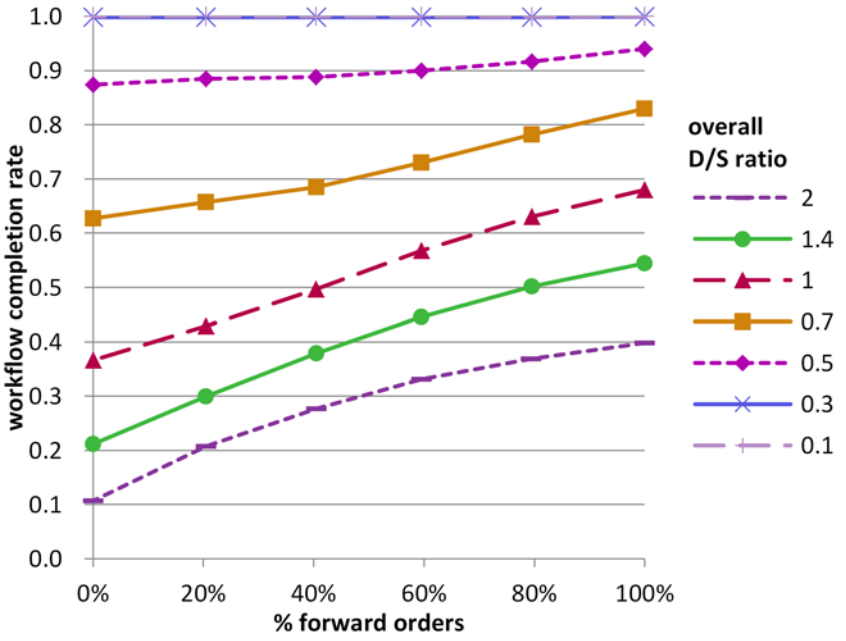


Figure 6.25 Workflow completion rate (dual market extensive)

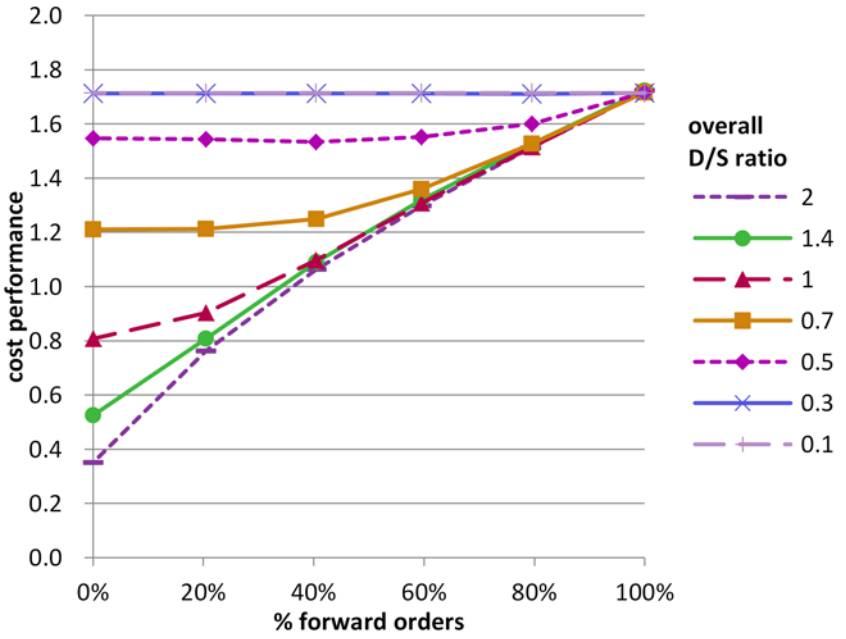


Figure 6.26 Cost performance (dual market extensive)

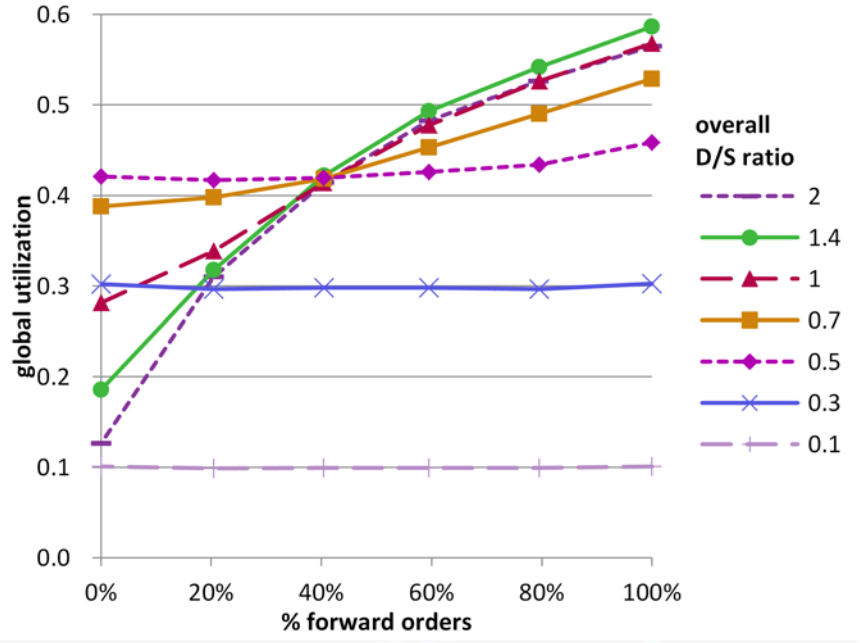


Figure 6.27 Global utilization (dual market extensive)

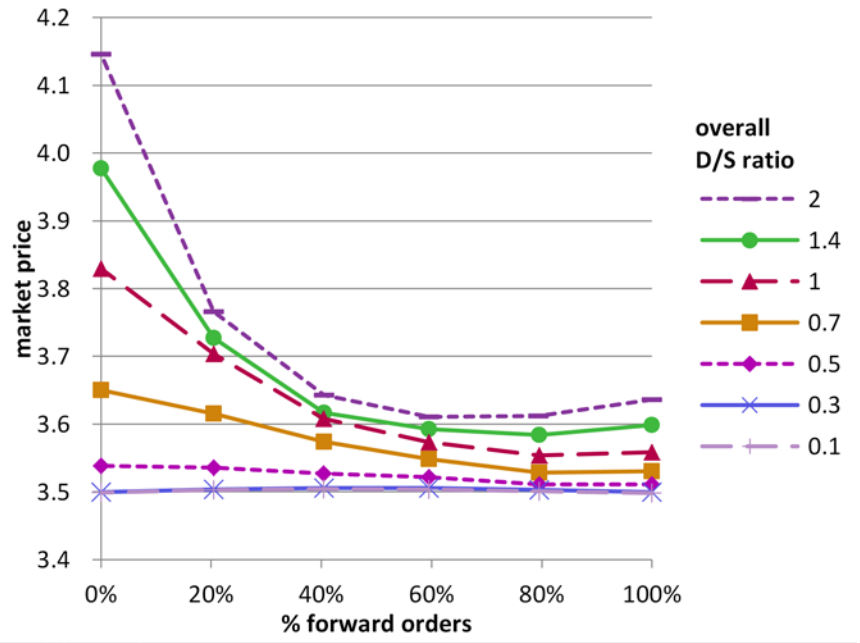


Figure 6.28 Market price (dual market extensive)

6.3.4 Summary

Three experiments have been carried out to evaluate the marketplace designs. The first experiment compared four types of market design running alone. The results showed that the forward/combinatorial design brings the best completion rate and cost performance for the users as well as the highest global utilization and second lowest market price. The second experiment confirmed that operating the spot market along with the forward market does not disturb the advance reservations while increasing the global utilization. The third experiment thoroughly simulated possible behaviors of the users. The results again clarified the advantage of the forward market against high demand/supply ratio. Although these results depend on the simulation settings, we can say that establishing a marketplace with not only the spot market but also the forward market can significantly improve its performance for both the users and the providers.

Additionally, the performance of conventional scheduling systems is estimated to be similar with the separate market mechanism, since no conventional scheduler is able to reserve a combination of resources for a workflow. The single-market experiment demonstrated that forward/combinatorial mechanism always outperforms the separate market mechanisms in WC and GU. Consequently, we can say that the proposed forward/combinatorial mechanism outperforms any conventional scheduling systems under the condition discussed in this thesis.

Chapter 7 Conclusions

7.1 Achievements

An intelligent marketplace for optimal service allocation in the cloud computing environment is desired to be established. This thesis proposed a combinatorial auction-based marketplace mechanism to support enterprise workflow-based applications built on the cloud services. It employs an exact optimization technique to achieve a Pareto-efficient allocation of services. The proposed marketplace consists of a forward market for an advance reservation and a spot market for immediate allocation of services so that the users reliably plan the use of cloud services within budget limitations.

Three experiments have been carried out in this thesis: (1) validation of combinatorial allocations, (2) estimation of mechanism overhead, and (3) evaluation of market performance. The first experiment demonstrated that the proposed mechanism succeeded to do matchmaking between multiple services and combinatorial requests according to the allocation scheme, as well as to determine their price according to the pricing scheme. The second experiment evaluated the overhead of matchmaking and pricing schemes of the proposed mechanism, and showed that the overhead is acceptable in an expected cloud computing environment. The third experiment compared four types of market mechanism, and showed that the proposed forward/combinatorial mechanism brought a superior

performance in both individual usability and system-wide efficiency. The third experiment also simulated a wide range of the users' preference between forward/spot markets, and concluded that the more users preferred the forward market, the more performance could be achieved against tight demand/supply conditions.

7.2 Future Works

Sophisticated strategies of seller/buyer agents can significantly improve the performance of the market. For instance, a buyer agent can reduce wasted resources and can increase workflow completion rate by employing a smarter strategy to make his orders. Moreover, it is essential for seller/buyer agents to adjust their order price according to the market price in competition with each other[6]. The future work will therefore investigate the market behavior using more sophisticated strategies of seller/buyer agents.

Interesting research objectives include the autonomous behavior of the market price, particularly the interaction between the forward market and the spot market, where the forward price is expected to be a forecast of the spot price. A high price in the forward market indicates a busy hour of the provider. By watching the forward price, a buyer can predict the demand-supply conditions and avoid purchasing an unnecessarily high-priced service. As a result, the demand-supply ratio (and thus the market price) is expected to be smooth and stable. This can be seen as an autonomous load-balancing functionality achieved by the dual-market design, which cannot be realized by the spot market alone. However, it is not clear that these

[6] The first fundamental theorem of welfare economics states that a competitive equilibrium among participants leads to a Pareto efficient allocation of resources [66]. This is also known as the invisible hand of God.

selfish participants are able to sustain such a peaceful marketplace; therefore further study with various agents is needed in the future research.

Real-world movements toward cloud computing marketplace are just ongoing. In May 2011, SpotCloud [64] opened a bilateral marketplace for both cloud service providers and consumers. The contribution of this thesis is not only to propose an efficient mechanism of cloud computing marketplace but also to encourage pioneers designing such a real-world marketplace. Our simulator, named W-Mart, can help them predict the behavior of the market by means of extensive multi-agent simulations. Its source code will be open to the public and contribute to further researches and developments.

7.3 Outlook

At the end of the thesis, let's imagine a world where the cloud service marketplace comes true…

The proposed market mechanism is so generic that any kind of service can be traded equally upon it. In a realistic scenario, however, the marketplace will be used hierarchically. For instance, a bundle of a low-level "storage service" and a high-level "corporate management consulting service" is not likely to be ordered, but that of a "storage service" and another low-level "networking service" are likely to be ordered. There appears a hierarchical structure within the marketplace.

Broker agents will play an important role in these hierarchical settings. A broker buys a bundle of some low-level services in the forward market, integrates them into a higher-level service, and sells it again in the spot market, typically at a higher price. For example, Dropbox [65] provides a high-level file synchronization/versioning service on top of Amazon's low-level virtual machine/storage service. As such, brokering is a major approach to do business in today's cloud computing

environment, and its importance will not change in future's cloud service marketplace. The proposed mechanism is excellent in that it supports a variety of brokering activities in a single marketplace.

The marketization of cloud computing services will raise a wide range of businesses and activities around the marketplace. It may include an insurance company who covers a failure of the service providers, a credit rating agency who evaluates the reliability and reputations of the providers, and an independent agency like the Securities and Exchange Commission who enforces fair trading on all the participants in the marketplace. The double-sided auction will certainly be the best mechanism to support fair trading among a variety of companies and citizens.

The author believes that it will be a keystone for a nation to have control over the cloud computing marketplace. The computing power is now fundamental to any business, science, government, military affairs, and social activities – we cannot live without them – and will be traded beyond the border as a commodity. Therefore, the meta-information generated by the marketplace (i.e. the worldwide flow/price/ demand/supply/etc. of the computing power) can indicate the movement of the world, and the quickest knowledge about it will bring an invaluable advantage to the nation who controls the marketplace.

References

[1] M. Armbrust, A. Fox, R. Griffith, A.D. Joseph, R.H. Katz, A. Konwinski, G. Lee, D.A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, *Above the Clouds : A Berkeley View of Cloud Computing*, 2009.

[2] "Microsoft Windows Azure" http://www.microsoft.com/windowsazure/.

[3] "Google App Engine" http://code.google.com/appengine/.

[4] I. Foster and C. Kesselman, "The grid: blueprint for a new computing infrastructure," Oct. 1998.

[5] S. Clearwater, *Market-Based Control: A Paradigm for Distributed Resource Allocation*, World Scientific, 1996.

[6] J. Shneidman, C. Ng, D.C. Parkes, A. AuYoung, A.C. Snoeren, A. Vahdat, and B. Chun, "Why Markets Could (But Don't Currently) Solve Resource Allocation Problems in Systems," *Challenges*, 2005, p. 7.

[7] P. Cramton, Y. Shoham, and R. Steinberg, *Combinatorial Auctions*, The MIT Press, 2005.

[8] A. Das and D. Grosu, "Combinatorial auction-based protocols for resource allocation in grids," *Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International*, 2005.

[9] A. Marshall, *Principles of Economics (Great Minds Series)*, Prometheus Books, 1997.

[10] "Market," *From Wikipedia, the free encyclopedia* http://en.wikipedia.org/wiki/Market.

[11] D. Friedman, "The double auction market institution: A survey," *The Double Auction Market: Institutions, Theories, And Evidence*, J. Rust, ed., Westview Press, 1993, pp. 3-26.

[12] R. Buyya, D. Abramson, J. Giddy, and H. Stockinger, "Economic models for resource management and scheduling in Grid computing," *Concurrency and Computation: Practice and Experience*, vol. 14, 2002, pp. 1507-1542.

[13] Y. Shoham and K. Leyton-Brown, *Multiagent systems: Algorithmic, game-theoretic, and logical foundations*, Cambridge Univ Pr, 2009.

[14] D.S. Diamond and L.L. Selwyn, "Considerations for computer utility pricing policies," *Proceedings of the 1968 23rd ACM national conference*, New York, New York, USA: ACM Press, 1968, pp. 189-200.

[15] I.E. Sutherland, "A futures market in computer time," *Communications of the ACM*, vol. 11, Jun. 1968, pp. 449-451.

[16] R. Buyya, D. Abramson, and S. Venugopal, "The Grid Economy," *Proceedings of the IEEE*, vol. 93, Mar. 2005, pp. 698-714.

[17] C.S. Yeo and R. Buyya, "A taxonomy of market-based resource management systems for utility-driven cluster computing," *Software: Practice and Experience*, vol. 36, Nov. 2006, pp. 1381-1419.

[18] R. Buyya and M. Murshed, "GridSim: a toolkit for the modeling and simulation of distributed resource management and scheduling for Grid computing," *Concurrency and Computation: Practice and Experience*, vol. 14, Nov. 2002, pp. 1175-1220.

[19] R. Buyya, R. Ranjan, and R.N. Calheiros, *Modeling and simulation of scalable Cloud computing environments and the CloudSim toolkit: Challenges and opportunities*, IEEE, 2009.

[20] C.A. Waldspurger, T. Hogg, B.A. Huberman, J.O. Kephart, and W.S. Stornetta, "Spawn: A Distributed Computational Economy," *IEEE Transactions on Software Engineering*, vol. 18, Feb. 1992, pp. 103-117.

[21] N. Nisan, S. London, O. Regev, and N. Camiel, "Globally distributed computation over the Internet-the POPCORN project," *Proceedings. 18th*

International Conference on Distributed Computing Systems, IEEE Comput. Soc, 1998, pp. 592-601.

[22] O. Regev, "The POPCORN market. Online markets for computational resources," *Decision Support Systems*, vol. 28, Mar. 2000, pp. 177-189.

[23] R. Wolski, J.S. Plank, T. Bryan, and J. Brevik, "G-commerce: market formulations controlling resource allocation on the computational grid," *Proceedings 15th International Parallel and Distributed Processing Symposium. IPDPS 2001*, IEEE Comput. Soc, 2001, p. 8.

[24] R. Wolski, J. Brevik, J.S. Plank, and T. Bryan, "Grid Resource Allocation and Control Using Computational Economies," *GRID COMPUTING: MAKING THE GLOBAL INFRASTRUCTURE A REALITY*, 2003, pp. 747 - 772.

[25] R. Buyya, D. Abramson, and J. Giddy, "Nimrod/G: an architecture for a resource management and scheduling system in a global computational grid," *Computer*, vol. 1, 2000, pp. 283-289.

[26] D. Abramson, I. Foster, J. Giddy, A. Lewis, R. Sosic, R. Sutherst, and N. White, "The Nimrod Computational Workbench: A Case Study in Desktop Metacomputing," *Australian Computer Science Communications*, vol. 19, 1997, pp. 17-26.

[27] I. Foster and C. Kesselman, "Globus: A Metacomputing Infrastructure Toolkit," *International Journal of Supercomputer Applications*, vol. 11, 1997, pp. 115-128.

[28] P. Padala, C. Harrison, N. Pelfort, E. Jansen, M.P. Frank, and C. Chokkareddy, "OCEAN: the open computation exchange and arbitration network, a market approach to meta computing," *Second International Symposium on Parallel and Distributed Computing, 2003. Proceedings.*, IEEE, 2003, pp. 185-192.

[29] A. AuYoung, B.N. Chun, A.C. Snoeren, and A. Vahdat, "Resource Allocation in Federated Distributed Computing Infrastructures," *Proceedings of the 1st Workshop on Operating System and Architectural Support for the Ondemand IT Infrastructure*, 2004, pp. 1-10.

[30] L. Peterson, T. Anderson, D. Culler, and T. Roscoe, "A blueprint for introducing disruptive technology into the Internet," *ACM SIGCOMM Computer Communication Review*, vol. 33, Jan. 2003, pp. 59-64.

[31] D. Oppenheimer, J. Albrecht, D. Patterson, and A. Vahdat, *Scalable wide-area resource discovery*, 2004.

[32] B.N. Chun, P. Buonadonna, A. AuYoung, D.C. Parkes, J. Shneidman, A.C. Snoeren, and A. Vahdat, "Mirage: A Microeconomic Resource Allocation System for Sensornet Testbeds," *The Second IEEE Workshop on Embedded Networked Sensors, 2005. EmNetS-II.*, IEEE, , pp. 19-28.

[33] K. Lai, L. Rasmusson, E. Adar, L. Zhang, and B.A. Huberman, "Tycoon: An implementation of a distributed, market-based resource allocation system," *Multiagent Grid Syst.*, vol. 1, 2005, pp. 169-182.

[34] T. Eymann, M. Reinicke, W. Streitberger, O. Rana, L. Joita, D. Neumann, B. Schnizler, D. Veit, O. Ardaiz, P. Chacin, I. Chao, F. Freitag, L. Navarro, M. Catalano, M. Gallegati, G. Giulioni, R.C. Schiaffino, and F. Zini, "Catallaxy-based Grid markets," *Multiagent and Grid Systems*, vol. 1, Dec. 2005, pp. 297-307.

[35] B. Schnizler, D. Neumann, D. Veit, M. Reinicke, and W. Streitberger, *Theoretical and Computational Basis for CATNETS - Annual Report Year 1*, 2005.

[36] D. Veit, G. Buss, B. Schnizler, and D. Neumann, *Theoretical and Computational Basis for CATNETS - Annual Report Year 2*, 2006.

[37] D. Veit, G. Buss, B. Schnizler, and D. Neumann, *Theoretical and Computational Basis for CATNETS - Annual Report Year 3*, 2007.

[38] Z. Tan and J.R. Gurd, "Market-based grid resource allocation using a stable continuous double auction," *Grid Computing, 2007 8th IEEE/ACM International Conference on*, 2007, pp. 283-290.

[39] T. Zhu and J.R. Gurd, "Market-based grid resource allocation using a stable continuous double auction (Ph.D. thesis)," 2007.

[40] "Amazon EC2 Spot Instances" http://aws.amazon.com/ec2/spot-instances/.

[41] "Heroku | Add-ons" http://addons.heroku.com/.

[42] "Google Apps Marketplace" http://www.google.com/enterprise/marketplace/.

[43] K. Hoki, "Outline of Japan Electric Power Exchange (JEPX)," *Transactions of the Institute of Electrical Engineers of Japan. B*, vol. 125, 2005, pp. 922-925.

[44] H. Kita, H. Sato, N. Mori, and I. Ono, "U-Mart system, software for open experiments of artificial market," *Computational Intelligence in Robotics and Automation 2003 Proceedings 2003 IEEE International Symposium on*, Ieee, 2003, pp. 1328 - 1333 vol.3.

[45] T.D. Cordeiro, D.B. Damalio, P.T. Endo, A.V. De Almeida Palhares, G.E. Gonçalves, D.F.H. Sadok, J. Kelner, B. Melander, V. Souza, and J.-E. Mångs, "Open Source Cloud Computing Platforms," *2010 Ninth International Conference on Grid and Cloud Computing*, 2010, pp. 366-371.

[46] P.T. Endo, G.E. Gonçalves, J. Kelner, and D. Sadok, "A Survey on Open-source Cloud Computing Solutions," *on Computer Networks*, 2009, pp. 3-16.

[47] "Cloud Foundry" http://cloudfoundry.com/.

[48] "OpenCloud" http://www.opencloud.com/.

[49] "Open Compute Project" http://opencompute.org/.

[50] "OpenStack" http://openstack.org/.

[51] "OpenShift" http://openshift.redhat.com/app/.

[52] "RightScale" http://www.rightscale.com/.

[53] "Engine Yard" http://www.engineyard.com/.

[54] "Heroku" http://www.heroku.com/.

[55] D. Amrhein, P. Anderson, A. de Andrade, J. Armstrong, E.A. B, R. Bruklis, K. Cameron, R. Cohen, A. Easton, R. Flores, G. Fourcade, T. Freund, B. Hosseinzadeh, W.J. Huie, P. Isom, S. Johnston, R. Kulkarni, A. Kunjunny, T. Lukasik, G. Mazzaferro, C. McClanahan, W. Melo, A. Monroy-Hernandez, D. Nicol, L. Noon, S. Padhy, G. Pfister, T. Plunkett, L. Qian, B. Ramachandran, J. Reed, G. Retana, D. Russell, K. Sankar, A.O. Sanz, W. Sinclair, E. Sliman, P. Stingley, R. Syputa, D. Tidwell, K. Walker, K. Williams, J.M. Willis, Y. Sasaki, E.

Windisch, and F. Zappert, *Cloud Computing Use Cases White Paper Version 2.0*, 2009.

[56] S. Weston, D. Green, G. Katsaros, T. Seed, N. Mc Donnell, and F. Scharinger, *GridCAE Grid for Computer Aided Engineering*, 2009.

[57] A. Mas-Colell, M.D. Whinston, and J.R. Green, "Microeconomic Theory," *The Canadian Journal of Economics*, vol. 21, 1995, p. 436.

[58] M. Satterthwaite and S. Williams, "The Bayesian theory of the k-double auction," *The Double Auction Market: Institutions, Theories, And Evidence*, J. Rust, ed., Westview, 1993, pp. 99-123.

[59] R.B. Myerson and M. Satterthwaite, "Efficient mechanisms for bilateral trading," *Journal of Economic Theory*, vol. 29, 1983, pp. 265-281.

[60] B. Schnizler, "MACE: A Multi-attribute Combinatorial Exchange," *Negotiation, Auctions, and Market Engineering*, 2008, pp. 84-100.

[61] "JOpt" www.eecs.harvard.edu/econcs/jopt/.

[62] "IBM ILOG CPLEX Optimizer" http://www.ibm.com/software/integration/optimization/cplex-optimizer/.

[63] "lp_solve" http://tech.groups.yahoo.com/group/lp_solve/.

[64] "SpotCloud" http://www.spotcloud.com/.

[65] "Dropbox" http://www.dropbox.com/.

[66] A.M. Feldman and R. Serrano, *Welfare economics and social choice theory*, Springer, 2006.

Publications

Journal Papers

[1] Ikki Fujiwara, Kento Aida, Isao Ono, "Combinatorial Auction-based Marketplace Mechanism for Cloud Service Reservation", *IEICE Transactions on Information and Systems*, E95-D(1), Jan 2012.

International Conferences

[2] Ikki Fujiwara, Kento Aida, Isao Ono, "Applying Double-sided Combinational Auctions to Resource Allocation in Cloud Computing", *Proceedings of the 10th Annual International Symposium on Applications and the Internet (SAINT2010)*, pp. 7-14, July 2010.

Refereed Domestic Conferences

[3] Ikki Fujiwara, Kento Aida and Isao Ono, "Market-based Resource Allocation for Distributed Computing", 情報処理学会 先進的計算基盤システムシンポジウム *SACSIS 2009*, pp. 149-150, May 2009.

Domestic Conferences

[4] 藤原一毅, 合田憲人, 小野功, "市場原理に基づく分散計算サービスの割り当て方法", ウィンターワークショップ 2011・イン・修善寺 論文集, 情報処理学会シンポジウムシリーズ *2011(2)*, pp. 67-68, January 2011.

[5] Ikki Fujiwara, Kento Aida and Isao Ono, "Market-based Resource Allocation for Distributed Computing", 2009 年並列／分散／協調処理に関する『仙台』サマー・ワークショップ (SWoPP 仙台 2009), 2009-HPC-121(34), August 2009.

[6] Ikki Fujiwara, Kento Aida and Isao Ono, "Market-based Service Allocation for Distributed Computing", 電子情報通信学会 第 13 回サイバーワールド研究会, pp. 35-38, June 2009.

Software

[7] "W-Mart source code repository" https://github.com/ikfj/wmart

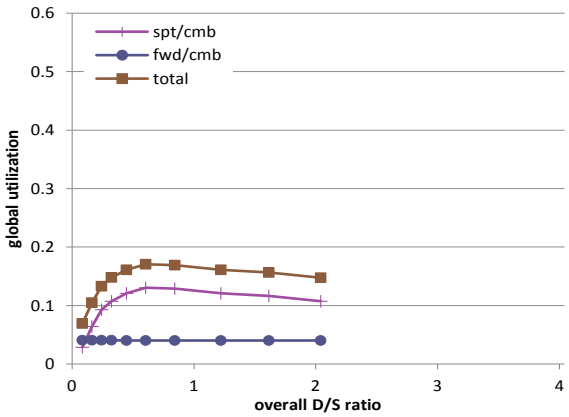
Appendix

A.1 Complete Results of Dual-market Basic Experiment

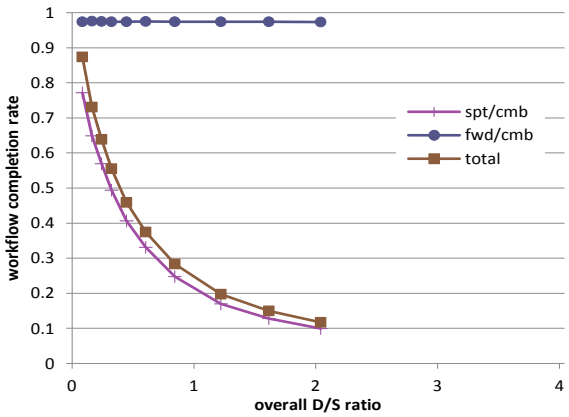
The supplemental results of the dual-market basic experiment (Section 6.3.3B) are shown in this appendix. In the following pages GU stands for the global utilization, WC stands for the workflow completion ratio, and CP stands for the cost performance.

D/S of *fwd/cmb* = 0.04

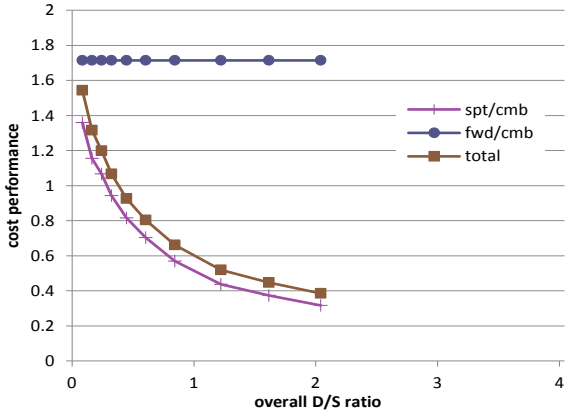
GU



WC

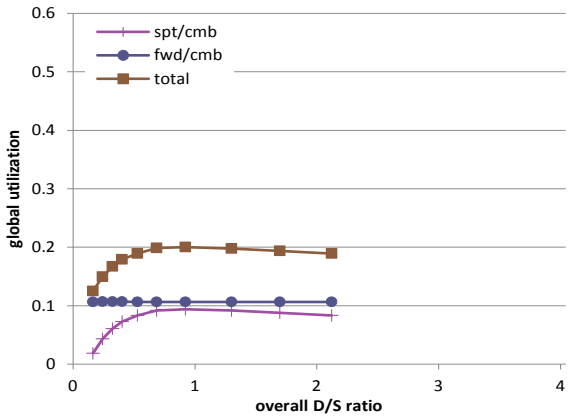


CP

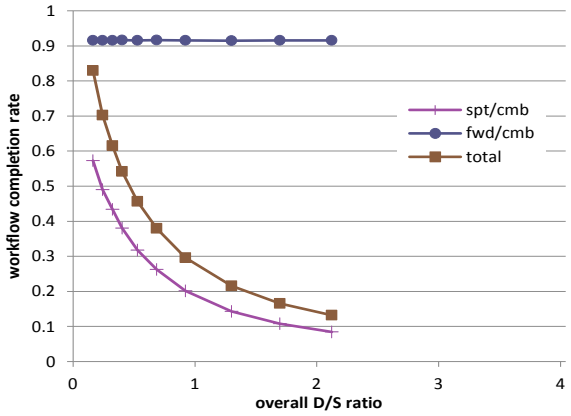


D/S of *fwd/cmb* = 0.12

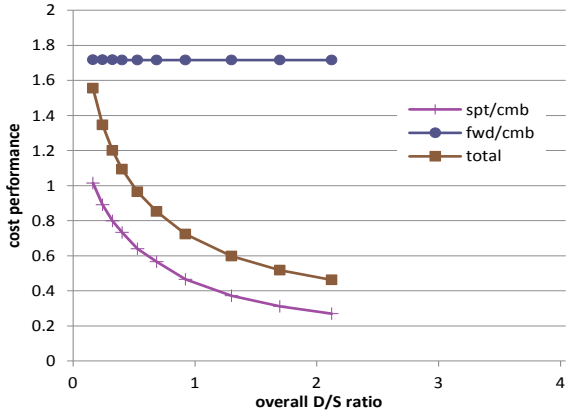
GU



WC

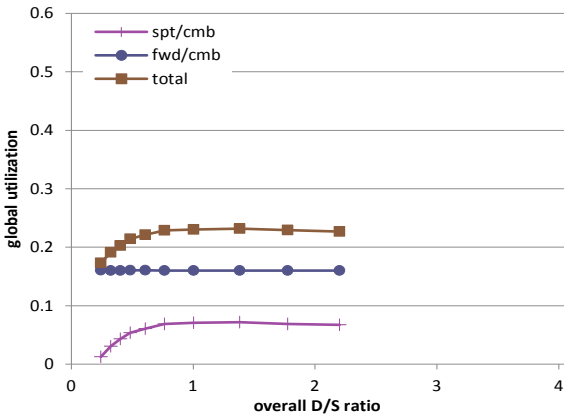


CP

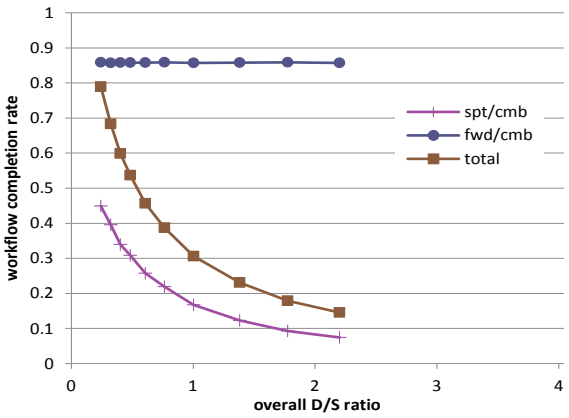


D/S of *fwd/cmb* = 0.20

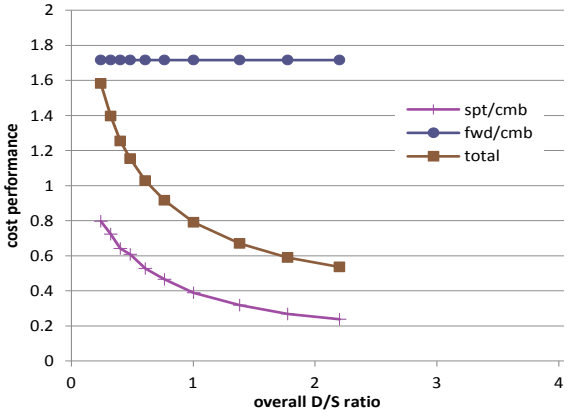
GU



WC

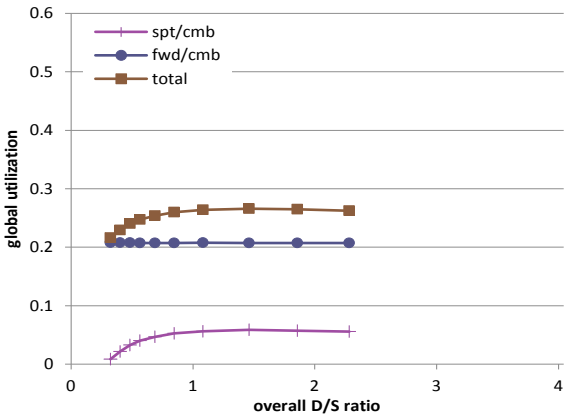


CP

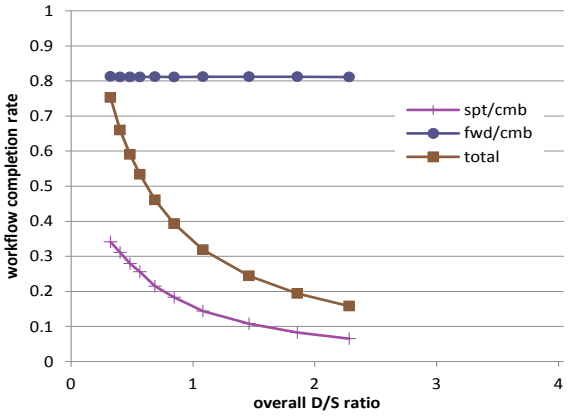


D/S of *fwd/cmb* = 0.28

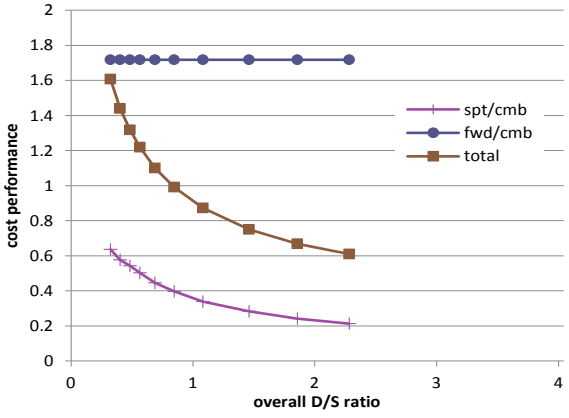
GU



WC

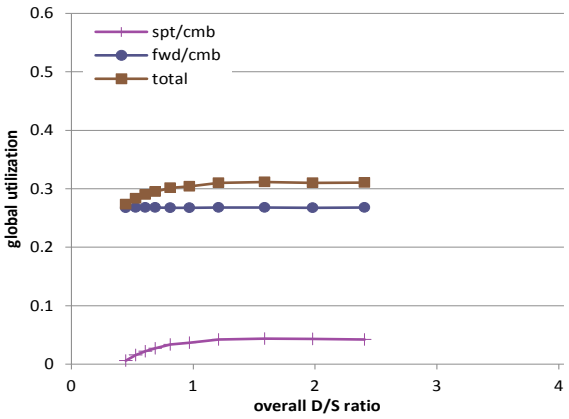


CP

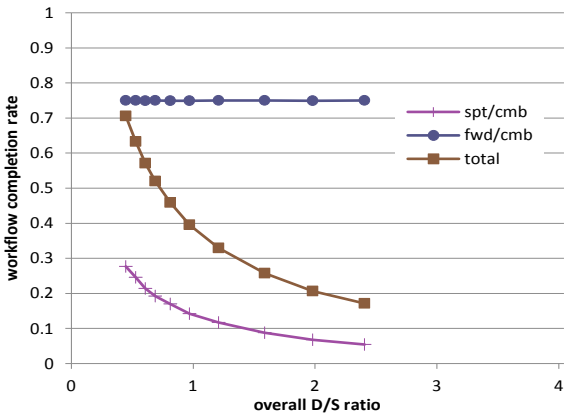


D/S of *fwd/cmb* = 0.40

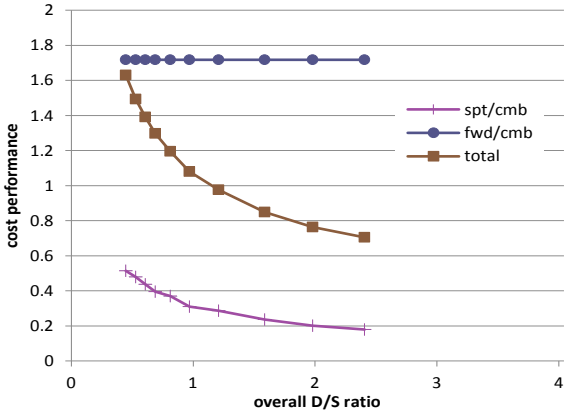
GU



WC

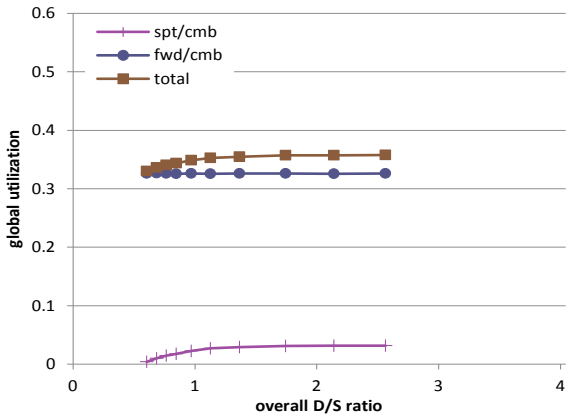


CP

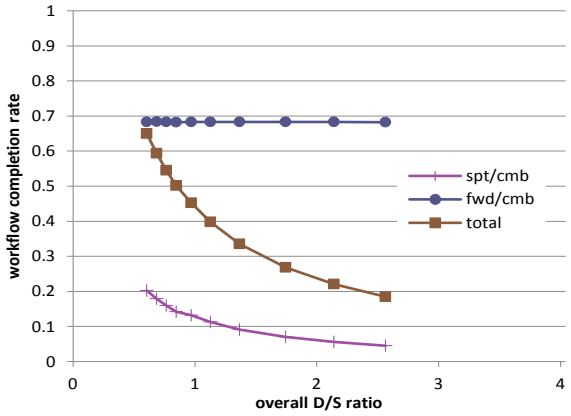


D/S of *fwd/cmb* = 0.56

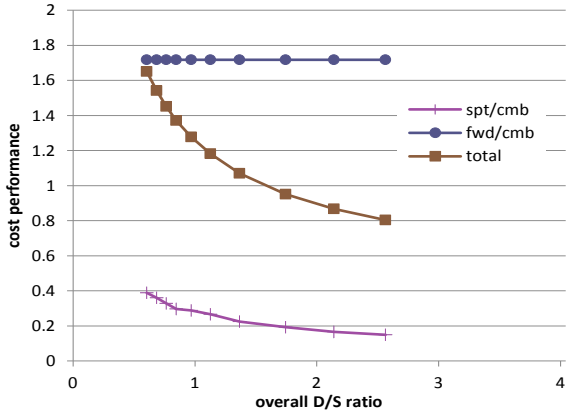
GU



WC

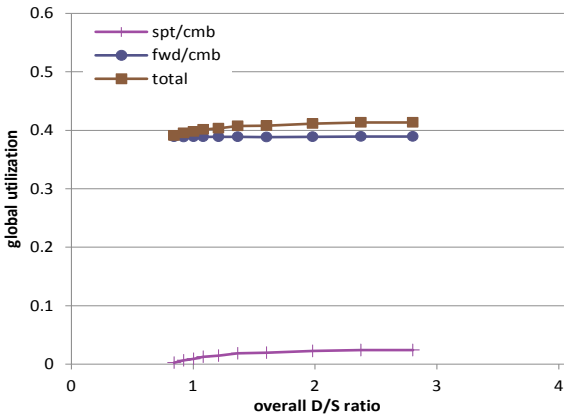


CP

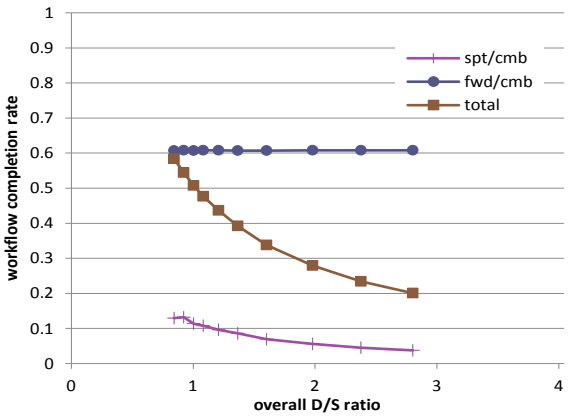


D/S of *fwd/cmb* = 0.80

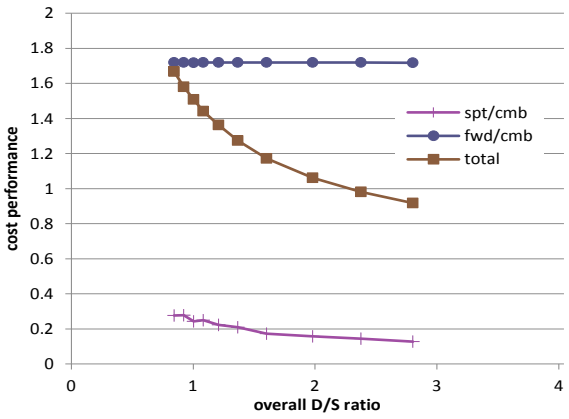
GU



WC

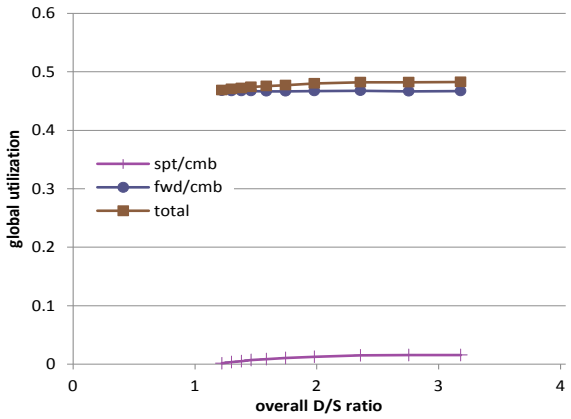


CP

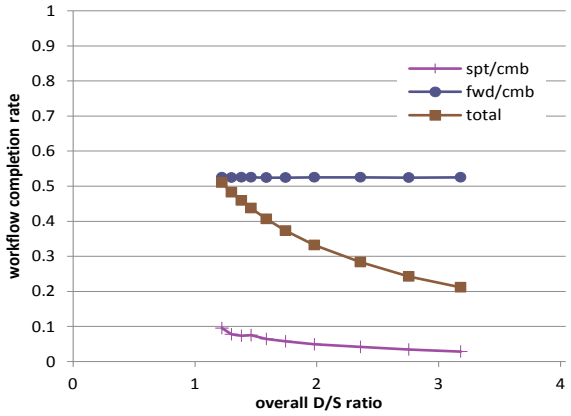


D/S of *fwd/cmb* = 1.18

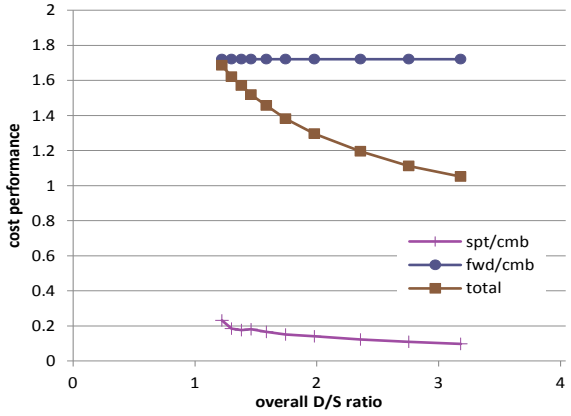
GU



WC

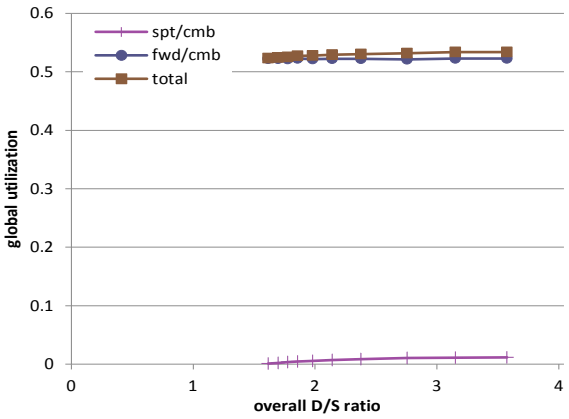


CP

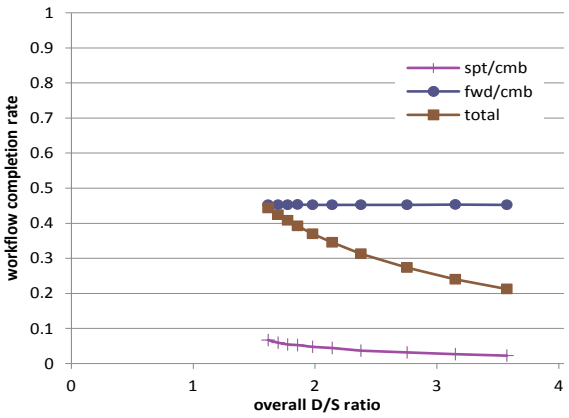


D/S of *fwd/cmb* = 1.56

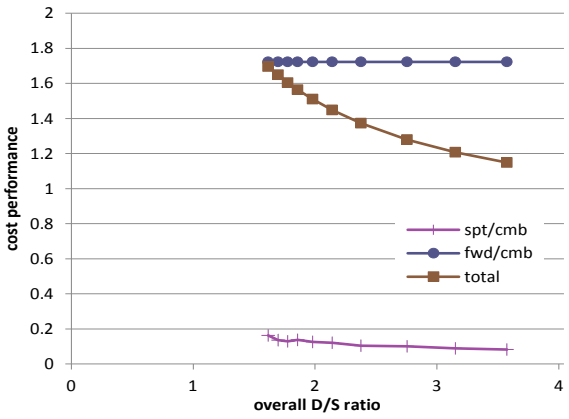
GU



WC

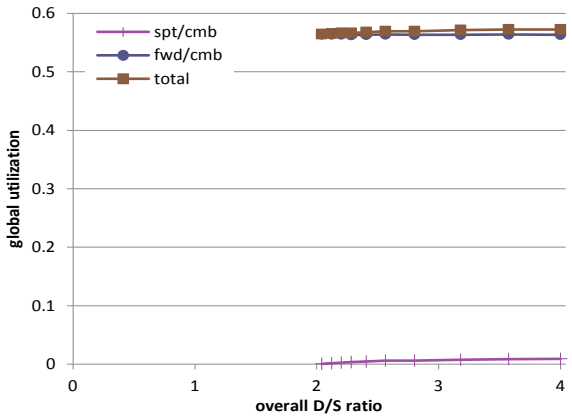


CP

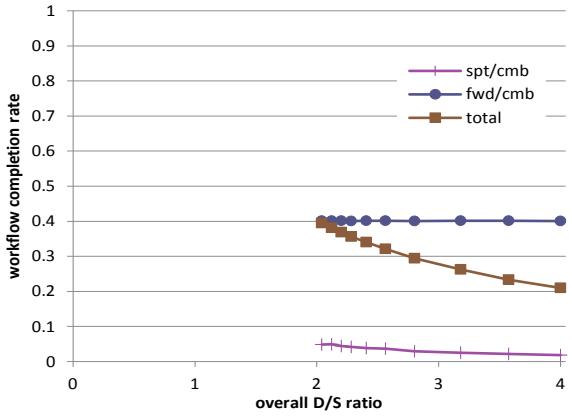


D/S of *fwd/cmb* = 2.00

GU



WC



CP