

# Tradeoffs between Profit and Customer Satisfaction for Service Provisioning in the Cloud

Junliang Chen<sup>\*</sup>  
Centre for Distributed and  
High Performance Computing  
School of Information  
Technologies  
The University of Sydney  
NSW 2006, Australia  
jchen@it.usyd.edu.au

Lei Sun  
School of Information and  
Electrical Engineering  
China University of Mining and  
Technology  
Xuzhou 221116, P.R. China  
leisuncumt@yahoo.com

Chen Wang  
CSIRO ICT Center  
PO Box 76, Epping  
NSW 1710, Australia  
chen.wang@csiro.au

Young Choon Lee  
Centre for Distributed and  
High Performance Computing  
School of Information  
Technologies  
The University of Sydney  
NSW 2006, Australia  
ycllee@it.usyd.edu.au

Bing Bing Zhou  
Centre for Distributed and  
High Performance Computing  
School of Information  
Technologies  
The University of Sydney  
NSW 2006, Australia  
bbz@it.usyd.edu.au

Albert Y. Zomaya  
Centre for Distributed and  
High Performance Computing  
School of Information  
Technologies  
The University of Sydney  
NSW 2006, Australia  
albert.zomaya@sydney.edu.au

## ABSTRACT

The recent cloud computing paradigm represents a trend of moving business applications to platforms run by parties located in different administrative domains. A cloud platform is often highly scalable and cost-effective through its pay-as-you-go pricing model. However, being shared by a large number of users, the running of applications in the platform faces higher performance uncertainty compared to a dedicated platform. Existing Service Level Agreements (SLAs) cannot sufficiently address the performance variation issue.

In this paper, we use utility theory leveraged from economics and develop a new utility model for measuring customer satisfaction in the cloud. Based on the utility model, we design a mechanism to support utility-based SLAs in order to balance the performance of applications and the cost of running them. We consider an infrastructure-as-a-service type cloud platform (e.g., Amazon EC2), where a business service provider leases virtual machine (VM) instances with spot prices from the cloud and gains revenue by serving its customers. Particularly, we investigate the interaction of service profit and customer satisfaction. In addition, we present two scheduling algorithms that can effectively bid for different types of VM instances to make tradeoffs be-

tween profit and customer satisfaction. We conduct extensive simulations based on the performance data of different types of Amazon EC2 instances and their price history. Our experimental results demonstrate that the algorithms perform well across the metrics of profit, customer satisfaction and instance utilization.

## Categories and Subject Descriptors

C.2.4 [Computer-Communication Networks]: Distributed Systems—*Distributed applications*; J.4 [Computer Applications]: Social and Behavioral Sciences—*Economics*

## General Terms

Design, Economics, Algorithms

## Keywords

utility theory, customer satisfaction, service level agreement, scheduling

## 1. INTRODUCTION

With the recent infrastructure-as-a-service (IaaS) computing paradigm, a business is enabled to run its online services on VM instances rented from one or multiple infrastructure service providers in a pay-as-you-go manner. The new computing style reduces the total cost of ownership (TCO) of online services by eliminating up-front investment and cutting down maintenance cost of hardware and software. An infrastructure service provider often provides a variety of virtualized resources to satisfy diverse business needs, e.g., Amazon Elastic Compute Cloud (EC2) offers VM instances that differ in computing/memory capacity, OS type, pricing scheme and geographic location for rent on an hourly basis [1]. Due to the intrinsic scalability and cost-effectiveness, it is becoming a trend to deploy services in the cloud [3].

<sup>\*</sup>The author is also with National ICT Australia Limited, 13 Garden Street, Australian Technology Park, Eveleigh, NSW 2015, Australia

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

HPDC'11, June 8–11, 2011, San Jose, California, USA.

Copyright 2011 ACM 978-1-4503-0552-5/11/06 ...\$10.00.

As shown in Figure 1, there are three typical parties in the cloud: infrastructure service providers, business service providers and customers (end users). Infrastructure service providers charge business service providers for renting VM instances to deploy service capacity. Business service providers charge customers for processing their requests. In the paper, we use business service provider and service provider interchangeably. These parties form a market in the cloud. Resource management in this market falls into two levels: the *infrastructure service provider level* and the *business service provider level*. The former mainly deals with the placement and migration of VM instances for the purpose of improving the efficiency of physical resource use, which has drawn considerable attention from both industry and academia. The latter deals with how a business service provider can efficiently use virtualized resources to achieve its business goals.

Compared with other markets, the cloud computing market faces challenges on providing certainty to the commodities exchanged among buyers and sellers. It has been noticed that performance variation of cloud computing nodes is high and some nodes may have order of magnitude worse performance than other nodes [5, 20]. It becomes a significant issue for an infrastructure service provider and a business service provider if they intend to offer viable service level agreements (SLAs) to their customers. Lack of such agreements may cause valuable applications to move away from the cloud and will jeopardize sustainable growth of cloud computing in the future.

To a certain degree, the problem arises due to the fact that each party in the cloud has its own interest. An infrastructure service provider intends to optimize its physical resource utilization for profit, which may affect the performance of VM instances running on it. On the other hand, a business service provider intends to satisfy its customers with minimal cost of renting VM instances. The utility model in economics is commonly used to represent the needs of a customer and the obligation of a provider. Under the

utility model, the prices of resources provided by the infrastructure service provider dynamically reflect the supply and demand of these resources. This approach has been known as utility computing [8, 10, 12, 18] for a while and is the cornerstone for resource allocation in the infrastructure service level. In this paper, we investigate the use of the utility model to characterize the SLA between a business service provider and its customers so that the performance of processing a request is proportional to its price. We investigate scheduling strategies in the business service provider level, particularly, we intend to optimize the strategies of choosing VM instances of different types and dispatching customer requests to these instances for achieving objectives on service profit and customer satisfaction.

Our work differs from previous studies in two aspects:

1. We consider customer satisfaction as an explicit metric when making schedules, while existing algorithms only focus on profit. Customer satisfaction is essential for building long-term profitable services in the business world. We believe it is also of foremost importance in characterizing an SLA and delivering services in the cloud.
2. Our scheduling algorithms deal with a varying number of resources with different types and prices, while most existing algorithms, such as FirstPrice [8], FirstReward [12] and FirstProfit [18] only apply to a fixed resource set. For scheduling in a fixed resource set, the key point is how to prioritize service requests that wait for processing. Given the fact that the number of resources in the cloud can be seen as unlimited from a business service provider's perspective, how to build an appropriate resource set according to the fluctuation of dynamic service request volume becomes a crucial issue. In many cases, two sets of resources may deliver different performance for different services, but they may have the same price.

The main contributions of our work include the following: 1) we develop a utility model using utility theory leveraged from economics for measuring customer satisfaction; 2) we investigate the impact of using different types of VM instances for request processing on the service provider's profit and the customer's satisfaction; 3) we show the interaction of service profit and customer satisfaction with our utility model; 4) we give two scheduling algorithms that can make tradeoffs between profit and customer satisfaction.

The rest of the paper is organized as follows: Section 2 gives an overview of the service provisioning problem we study in this work; Section 3 presents the utility model and a mechanism to support utility-based SLAs; Section 4 describes our scheduling algorithms; Section 5 presents the Amazon EC2 based evaluation results of our algorithms; Section 6 is the related work and Section 7 concludes the paper.

## 2. THE SYSTEM MODEL

We consider a business service that delivers application services on demand in the cloud. The applications can be scientific data processing, finance data analysis, image processing, video encoding, etc. The service provider charges customers who request to run these applications. We use

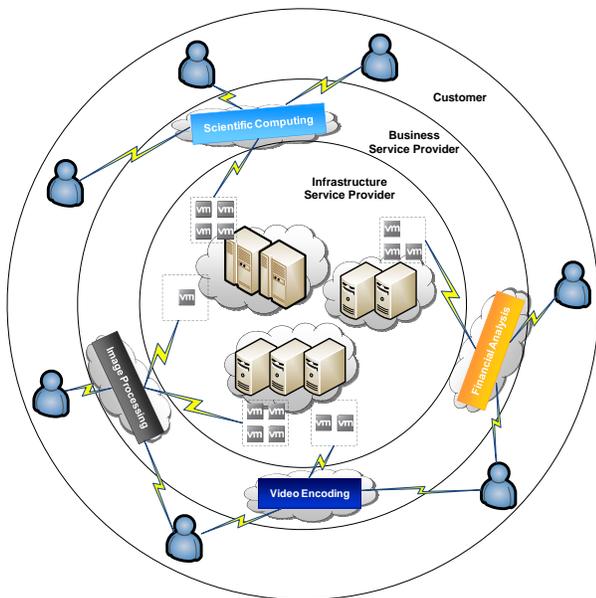


Figure 1: The three-tier cloud computing market

**Table 1: The configurations of three types of spot instances on Amazon (California, US, Jan 2011)**

Instance type	CPU (core)	Memory (GB)	Storage (GB)
Small	1	1.7	160
Large	2	7.5	850
Extra Large	4	15	1690

**Table 2: Results of encoding 512-frame 1080p video streams using x264 on Amazon EC2**

Instance type	mean (sec)	standard deviation (sec)
Small	402.9	4.9
Large	101.2	1.6
Extra Large	56.6	1.0

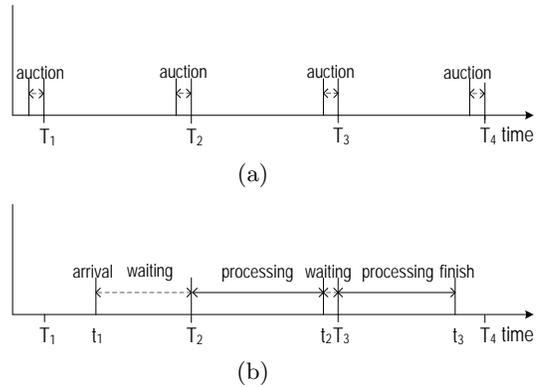
x264 [4] as a sample application to show its performance difference on several types of VM instances on Amazon EC2. X264 is a free software tool for encoding video streams into H.264/MPEG-4 AVC format. It is used in many Web video services such as Youtube and Facebook. It is capable of performing video encoding tasks on both single and multiple processor machines. The execution time of a video encoding task may vary from several minutes to many hours, depending on the video size. We run x264 on Amazon EC2 using three types of VM instances from the spot instance pool. The configuration of the spot instances is shown in Table 1. Table 2 lists the average execution time and variation of 10 runs of encoding 512-frame 1080p video streams on different types of instances using x264. The problem for the service provider is how to rent a set of VM instances from an infrastructure service provider to serve customer requests so that tradeoffs between its profit and customer satisfaction can be achieved.

We consider the problem under the following system model.

A service provider rents VM instances through participating auctions run by the infrastructure service provider. The auctions are held in fixed time intervals using an algorithm similar to ascending clock auction [22]. The results of an auction set the prices of instance types and reflect the current supply and demand of these instance types. At the end of each time interval, an auction is held to take users' bids. The prices of different types of instances for next time interval are then determined. If bidders accept the updated prices, instances that they bid for will be launched at the beginning of next time interval. Incurred resource usage is charged at the end of each time interval. The minimum unit of charge is a time interval, partial resource usage that is less than a time interval will be billed as a full time interval. The infrastructure service provider model used in this paper is similar to Amazon EC2 Spot Instances [2].

Customers of the service provider submit service requests to run applications. A request  $r_j$  is characterized by a four-parameter tuple  $(size_j, U_{0j}, \alpha_j, \beta_j)$ , where  $size_j$  is the request size (similar to the job size);  $U_{0j}$  is the maximum utility the required service delivers to the customer;  $\alpha_j$  and  $\beta_j$  denote the customer's preference for service price and response time. We will discuss this in detail later in Section 3.

The scheduling of these requests is performed in a semi-online manner. The service provider buffers service requests that arrive in current time interval in a queue, and bids for instances to process them in the next time interval. We assume that the time interval between two auctions is smaller



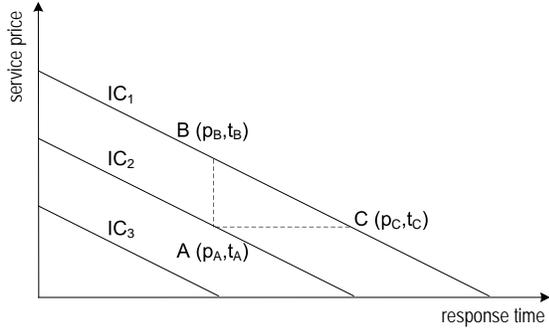
**Figure 2: An example of the scheduling process. (a) auction cycles; (b) the scheduling process of a request.**

than the average request size so that the time of waiting is not significant. When the auction opens for bids at the end of current time interval, the service provider runs a scheduling algorithm to bid for instances. Once the VM instances are successfully bid and the next time interval starts, the service provider dispatches the buffered requests to the instances. The scheduling algorithm consists of two steps. In the first step, it determines how many instances of different types to bid for according to the information of requests in the queue (incentives of customers) and the market prices of different types of instances (constraints of the infrastructure service provider). The service provider needs to rent a set of instances to maximize its profit or improve customer satisfaction. In the second step, the scheduling algorithm assigns requests to rented instances. If the processing of a request cannot finish in the current time interval, the request will stay in the queue and join the auction for the next time interval. Due to the fluctuations of instance prices, the instance for processing the request in the next time interval can be of a different type. Here we assume that the request is divisible and moving the remainder of a request to another instance does not incur much cost. The assumption is realistic as many instances rented by the service provider may share the same file system.

We illustrate the scheduling process through an example. As shown in Figure 2(a),  $T_1, T_2, T_3$  and  $T_4$  are beginnings of four time intervals, auctions are held at the end of each time interval. In Figure 2(b), a request  $r$  arrives at time  $t_1$ , then it is put into the waiting queue. At time  $T_2$ , the start of the next time interval,  $r$  is scheduled to be processed on an instance. At time  $t_2$ , the intermediate result of  $r$  is saved and it is put back into the queue. The request is rescheduled to run at time  $T_3$  on an instance that may or may not be the previous one. Finally, the request processing finishes at time  $t_3$ . The response time of request  $r$  is  $t_3 - t_1$ , and the delay is  $T_2 - t_1$ .

### 3. THE UTILITY MODEL AND UTILITY-BASED SLAS

Previous research on market-based resource allocation often only takes into account the values of customer requests, but fails to characterize the satisfaction of these customers. Customer satisfaction measures how well service performance



**Figure 3: An example of indifference map.**  $IC_1$ ,  $IC_2$  and  $IC_3$  are three indifference curves with satisfaction level  $U_1$ ,  $U_2$  and  $U_3$  respectively ( $U_1 < U_2 < U_3 < U_0$ ). A, B and C are points on the indifference curves, representing different combinations of service price and response time.

meets customer expectation. It is a key indicator in the business world.

Previous work [8, 12, 18] considers that the value of a customer request changes with the time it stays in a system. There is no indication about how a customer satisfies with different combinations of service price and request processing time. We model the customer satisfaction based on the utility theory [17] in economics. As shown in Equation 1, we define the satisfaction, or utility of using a service as a function of the service price  $p$  and the response time  $t$ .

$$U(p, t) = U_0 - \alpha p - \beta t \quad (1)$$

in which,  $U_0$  is the maximum utility that the service delivers to the customer;  $\alpha$  and  $\beta$  are coefficients.  $U_0$  is proportional to the size of the service request. It is natural that serving a request of large size means high utility for a customer.  $\alpha/\beta$  ( $\beta/\alpha$ ) is known as marginal rate of substitution in economics, denoting the rate at which the customer is willing to give up response time (or service price) in exchange for service price (or response time) without any satisfaction change.

A customer may have different levels of satisfaction. An indifference map [17] can be drawn based on satisfaction levels, as shown in Figure 3. Each line (a special case of indifference curve) in the map represents a certain level of satisfaction. The points on a given line represent different combinations of service price and response time that deliver the same satisfaction level, i.e., the customer is equally satisfied at all points on a given line. However, the customer prefers lines with higher satisfaction levels to those with lower satisfaction levels. For example, as illustrated in Figure 3, the customer has no preference between points B and C, but prefers point A to points B and C.

Equation 1 enables us to compare customer satisfaction levels as long as the price and response time of the request processing are known. In addition, when fixing the satisfaction to a customized level  $U_c$  ( $U_c < U_0$ ), the service price  $p$  that the customer would like to pay for request processing can be calculated as below:

$$p = \frac{U_0 - U_c - \beta t}{\alpha} \quad (2)$$

The utility model further enables us to define utility-based SLAs between a customer and a service provider. An SLA in this case is the tuple  $(U_0, \alpha, \beta)$ , which constrains how the service performance is satisfactory to a customer. Apparently, the relationship between the price and response time determines the satisfaction.

With a set of SLAs between a service provider and a number of customers, the service provider is capable of making efficient decisions regarding VM instance renting to satisfy both its own profit target and customers' needs. The flexibility in price and response time can also relieve the performance variation issue in the cloud. Specially, a service provider can do the following to optimize its resource use:

1. While maintaining a certain level of customer satisfaction, the service provider is enabled to optimize profit by reducing the response time and charging a higher service price, which means moving the point  $(p, t)$  up left along the indifference curve, e.g., from point C to point B in Figure 3.
2. While keeping a profit target, the service provider can improve customer satisfaction by reducing the response time, which means moving point  $(p, t)$  left horizontally from an indifference curve to another indifference curve with higher satisfaction level, e.g., from point C to A in Figure 3.
3. For delayed services caused by performance variation of rented VM instances, the service provider can maintain the customer satisfaction by charging a lower service price, which means moving the point  $(p, t)$  down right along the indifference curve, e.g., from point B to point C in Figure 3.

We will describe two algorithms to achieve these in the next section.

## 4. SCHEDULING ALGORITHMS

Our scheduling algorithms are based on the model and SLAs described above. In this section, we first introduce *performance index (PI)* to measure the performance differences of different instance types. We then describe our portfolio strategies for building an appropriate resource set to process service requests in a semi-online manner. Finally, we describe two scheduling algorithms for making tradeoffs between profit and customer satisfaction.

### 4.1 Performance Index

Processing a request on VM instances of different types may have great difference in response time, and hence the service provider's profit and customer satisfaction (see Table 2). To quantify the performance difference, we normalize the request processing capacity of VM instances against that of a standard instance. We call the normalized capacity *performance index*. Let  $w_0$  and  $w_k$  denote the workload that a standard instance and a type  $i_k$  instance can process in a time interval respectively. The performance index, denoted by  $PI_k$  is defined as below:

$$PI_k = \frac{w_k}{w_0} \quad (3)$$

Suppose a standard instance uses time  $t_0$  to process a request, a type  $i_k$  instance shall normally need time  $\frac{t_0}{PI_k}$

to process the same request. A service provider can obtain performance indexes of various instance types for processing requests through profiling and benchmarking.

## 4.2 Portfolio Strategies

We use  $I = \{i_1, \dots, i_m\}$  to denote the set of instance types and  $R = \{r_1, \dots, r_n\}$  to denote the requests in a waiting queue attached to a service provider. For each request  $r_j \in R$ , the following variables are defined to describe its state:

- $cost_j$ : the accumulated cost of instance renting for processing  $r_j$ .  $cost_j$  is updated every time interval because the cost of instance renting is charged per time interval by the infrastructure service provider.
- $revenue_j$ : the revenue that a service provider expects to generate by serving  $r_j$ . The revenue is realized only when  $r_j$  is finished processing, i.e., the service provider charges the customer only when his/her request is finished (not every time interval).
- $rpt_j$ : the remaining processing time (on a standard instance) of  $r_j$ . It is also updated every time interval. The initial value of  $rpt_j$  equals the request size  $size_j$ .

As mentioned earlier, at the end of each time interval, a service provider needs to bid for VM instances to process service requests in the queue for the next time interval. The service provider makes decisions on what types of instances and how many instances to bid for. Since performance and prices of different types of instances vary greatly, processing a request on different types of VM instances results in different revenue and cost, hence different profit and customer satisfaction. It is of the service provider's interest to schedule requests to instance types so that its objective of maximizing profit or improving customer satisfaction is met. While profit is often the driving force, customer satisfaction can sometimes be a more preferable goal to pursue for a service provider as satisfaction may lead to competitiveness and long term profit, particularly when many service providers offer equivalent services in the cloud.

Based on the utility model and performance indexes of various instance types above, we develop portfolio strategies for a service provider to rent an appropriate set of VM instances to serve its customers. Portfolio strategy [6] is broadly used in financial investment to reduce price fluctuation risk by building an appropriate set of different assets, e.g., stocks, bonds, options. When deciding which type of instance to choose for processing a request in an auction session, our strategies calculate expected profit (or satisfaction) for different types of VM instances, and then choose the type with maximum expected profit (or satisfaction). Due to the price fluctuations, the instance type chosen for processing the same request may be different in different auction sessions.

### 4.2.1 Profit Optimization under a Satisfaction Target

This strategy is for service providers who aim to maximize profit while maintaining a minimal satisfaction level, denoted by  $U_{min}$  ( $U_{min} < U_0$ ).

In an auction session, for each request  $r_j$  in the queue, if it is scheduled to an instance of type  $i_k \in I$ , its expected remaining processing time would be as below:

$$rpt_{jk} = \frac{rpt_j}{PI_k} \quad (4)$$

The expected accumulated cost can be calculated using the following equation:

$$cost_{jk} = cost_j + rpt_{jk} \cdot p_k \quad (5)$$

where  $p_k$  is the market price of instance type  $i_k$  for the next time interval.

The expected revenue  $revenue_{jk}$  can then be calculated through Equation 2 with given  $U_{min}$  and the expected response time  $resp\_time_{jk}$ , which is derived from

$$resp\_time_{jk} = current\_time - arrival\_time_j + rpt_{jk} \quad (6)$$

where the two self-explaining parameters  $current\_time$  and  $arrival\_time_j$  denote the current time and the arrival time of request  $r_j$ .

The expected profit generated through processing  $r_j$  on an instance of type  $i_k$  is therefore as below:

$$profit_{jk} = revenue_{jk} - cost_{jk} \quad (7)$$

Among all instance types ( $k = 1, \dots, m$ ), the instance type with maximum expected profit will be selected for processing request  $r_j$ .

---

### Algorithm 1: FirstFit-profit algorithm

---

**Input:** market prices  $p_1, \dots, p_m, U_{min}$

- 1: **for** each request  $r_j \in R$  **do**
- 2:   Update  $rpt_j, cost_j$
- 3:    $profit_j = 0, instance_j = 0$
- 4:   **for** each instance type  $i_k \in I$  **do**
- 5:      $rpt_{jk} = \frac{rpt_j}{PI_k}$
- 6:      $cost_{jk} = cost_j + rpt_{jk} \cdot p_k$
- 7:     Calculate  $revenue_{jk}$  with  $U_{min}$  and  $resp\_time_{jk}$
- 8:      $profit_{jk} = revenue_{jk} - cost_{jk}$
- 9:     **if**  $profit_j < profit_{jk}$  **then**
- 10:        $profit_j = profit_{jk}$
- 11:        $instance_j = i_k$
- 12:     **end if**
- 13:   **end for**
- 14:   bid for an instance of type  $instance_j$  for processing request  $r_j$
- 15: **end for**

---

### 4.2.2 Satisfaction Optimization with a Profit Bound

This strategy is for service providers who aim to maintain a minimal unit profit  $profit_{min}$  for each request. Unit profit is defined as  $\frac{profit}{request\_size}$ .

In an auction session, if request  $r_j$  is scheduled to an instance of type  $i_k$ , the expected remaining processing time and expected accumulated cost are also calculated using Equation 4 and Equation 5.

The expected revenue under given  $profit_{min}$  is calculated as below:

$$revenue_{jk} = profit_{min} \cdot size_j + cost_{jk} \quad (8)$$

The expected satisfaction  $satisfaction_{jk}$  can therefore be calculated using Equation 1 with  $revenue_{jk}$  and  $resp\_time_{jk}$  (also derived from Equation 6).

Among all instance types, the instance type with maximum satisfaction will be selected for processing request  $r_j$ .

---

**Algorithm 2:** FirstFit-satisfaction algorithm

---

**Input:** market prices  $p_1, \dots, p_m, profit_{min}$

- 1: **for** each request  $r_j \in R$  **do**
- 2:   Update  $rpt_j, cost_j$
- 3:    $satisfaction_j = 0, instance_j = 0$
- 4:   **for** each instance type  $i_k \in I$  **do**
- 5:      $rpt_{jk} = \frac{rpt_j}{PI_k}$
- 6:      $cost_{jk} = cost_j + rpt_{jk} \cdot p_k$
- 7:      $revenue_{jk} = profit_{min} \cdot size_j + cost_{jk}$
- 8:     Calculate  $satisfaction_{jk}$  with  $revenue_{jk}$  and  $resp\_time_{jk}$
- 9:     **if**  $satisfaction_j < satisfaction_{jk}$  **then**
- 10:        $satisfaction_j = satisfaction_{jk}$
- 11:        $instance_j = i_k$
- 12:     **end if**
- 13:   **end for**
- 14:   bid for an instance of type  $instance_j$  for processing request  $r_j$
- 15: **end for**

---

### 4.3 Scheduling Algorithms

With the portfolio strategies, two scheduling algorithms naturally follow. FirstFit-profit algorithm is to maximize profit when maintaining a target of customer satisfaction, while FirstFit-satisfaction algorithm is to maximize customer satisfaction while keeping a bound of unit profit. These scheduling algorithms are invoked when a service provider has pending requests in its waiting queue during auction sessions.

## 5. PERFORMANCE EVALUATION

We evaluate our algorithms through extensive simulation based on the performance data of different types of Amazon EC2 instances and their price history.

In the experiments, we first investigate the impact of using instances of different types for request processing on the service provider’s profit. Specifically, we compare our algorithms with four baseline algorithms that use homogeneous instances, BL-small, BL-large, BL-xlarge and BL-random. The first three algorithms always choose *small*, *large* and *extra large* instances to rent respectively during an auction session whereas the last algorithm randomly chooses one from the three instance types to bid for. We then examine the performance of our scheduling algorithms with different customer preferences. At last, we investigate the relationship between service profit and customer satisfaction, and demonstrate the benefit a service provider may gain with a flexible satisfaction level (or profit bound).

### 5.1 Experimental Setting

We use three types of spot instances on Amazon in our experiments, i.e., *small*, *large* and *extra large*. Table 1 gives the configuration of these instance types. We run performance testing with x264 on instances of the three types to obtain their performance indexes. We also conduct performance testing with other applications such as Black-scholes and Postmark. The results are not shown here as the focus of this paper is not on handling applications with different types. We use the price history of these three instance types as the clearing prices of auctions of VM instances. The price history of the three VM instance types used in our exper-

**Table 3: Parameters used in the simulation**

Parameter	Value
Number of runs	10
Number of requests	10,000
Request arrival rate $\lambda$	15 per time interval
Minimum request size	2 time intervals
Maximum request size	50 time intervals
Maximum utility $U_0$	equals request size
$\alpha/\beta$	9, 3, 2, 1, 1/2, 1/4, 1/8
Instance types	<i>small</i> , <i>large</i> , <i>extra large</i>
Instance prices	Amazon spot instances price history

iments is shown in Figure 4. In our simulation, auctions open at the end of each time interval, during which prices of different instance types for next time interval are updated from price traces.

The simulation consists of 10 runs. Each run handles 10,000 customer requests that arrive in a Poisson process with rate  $\lambda$ . Each request is characterized by a request size (request processing time) and a utility function. The sizes of requests follow a bounded Pareto distribution. The minimum request size is set to 2 time intervals, while the maximum size is set to 50 time intervals. For the utility function,  $U_0$  is set to equal the request size, while  $\alpha/\beta$  has different settings in different experiments. As mentioned earlier,  $\alpha/\beta$ , marginal rate of substitution, denotes the customer’s preference over different combinations of price and response time. In the first and third experiments,  $\alpha/\beta$  is randomly selected from 9, 3, 2, 1, 1/2, 1/4 and 1/8 for each request, for the purpose of examining the performance of scheduling algorithms under multiple customer types. In the second experiment,  $\alpha/\beta$  is set to 9, 3, 1 for all requests respectively in order to examine the impact of customer preference on service profit, customer satisfaction and instance utilization. Table 3 lists the parameters used in our experiments.

We use the following performance metrics to evaluate our algorithms:

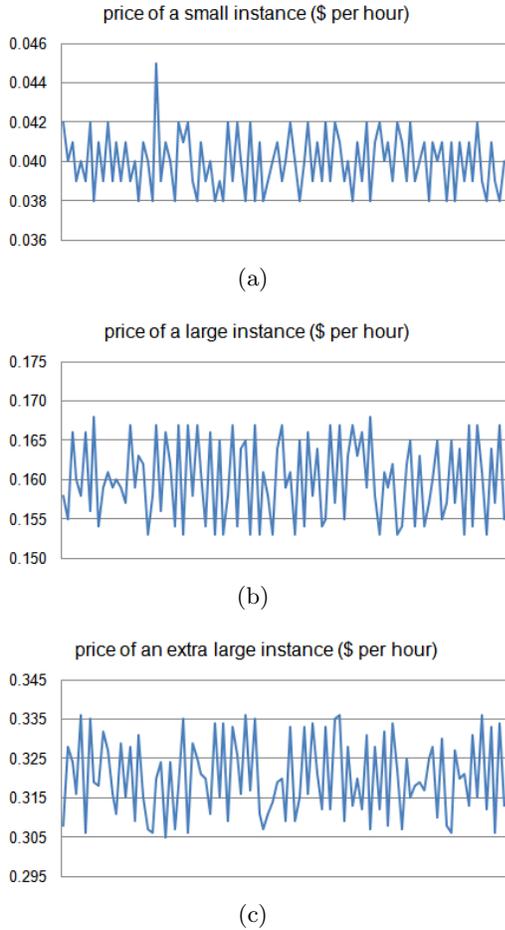
- Average unit profit (*profit*): the average unit profit gained by processing a request. *profit* is defined as below:

$$profit = \frac{\sum_{j=1}^n (\frac{revenue_j - cost_j}{size_j})}{n} \quad (9)$$

- Profit loss rate (*loss<sub>profit</sub>*): the fraction of requests that fail to make profit due to the fact that the revenue brought by them cannot cover the cost of renting instances to process them.
- Average satisfaction (*satisfaction*): the average customer satisfaction of processing a request. *satisfaction* is defined as the following:

$$satisfaction = \frac{\sum_{j=1}^n satisfaction_j}{n} \quad (10)$$

- Satisfaction loss rate (*loss<sub>satisfaction</sub>*): the fraction of requests that their associated customer satisfaction falls below 0.
- Number of instances (*number*): the average number of instances rented in a time interval.
- Utilization rate (*utilization*): the proportion of time that instances are busy processing requests.



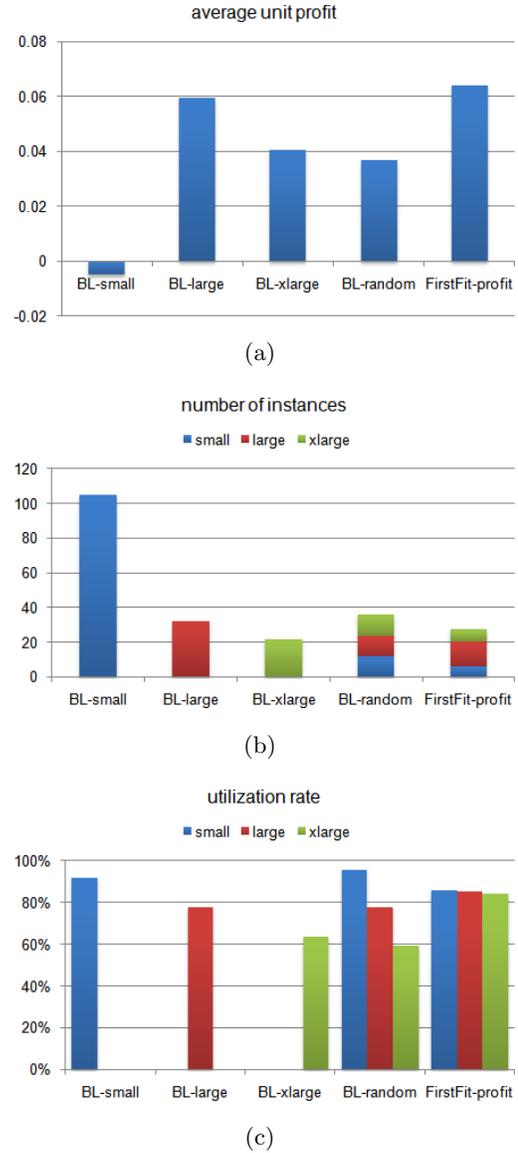
**Figure 4: Price history of three types of spot instances on Amazon (Linux, California, US, Jan 1 - Jan 15, 2011). (a) small instance; (b) large instance; (c) extra large instance.**

## 5.2 Results

In this section, we provide and analyze the experimental results. First, we evaluate the effectiveness of using VM instances of different types for service request processing by comparing with algorithms that use homogeneous instances. Second, we present the results of our scheduling algorithms on handling different customer types defined by different  $\alpha/\beta$  ratios. Third, we show the performance of our algorithms under different satisfaction targets and profit bounds to examine the relationship of service profit and customer satisfaction.

### 5.2.1 The effectiveness of using instances of different types

We measure the average unit profit gained through our FirstFit-profit algorithm and compare it with the average unit profit gained through BL-small, BL-large, BL-xlarge and BL-random in Figure 5(a). The FirstFit-profit algorithm outperforms all the algorithms that use instances of a single type. It also outperforms BL-random, which randomly selects different instance types, by a wide margin. To a great extent, it can be seen from the results that a service



**Figure 5: Performance of different scheduling algorithms (with satisfaction target 1.0). (a) average unit profit (\$ per time interval); (b) number of instances; (c) utilization rate.**

provider's profit is influenced by the way it selects instances. Our algorithm takes into account performance differences and price fluctuations of different instance types when making VM instance renting and request scheduling decisions. Those baseline algorithms, like many existing utility based algorithms, do not consider these factors.

Figure 5(b) shows the number of instances used by each algorithm and Figure 5(c) shows the utilization of these instances. It is clear that FirstFit-profit is capable of building a pool of instances that is just sufficient to handle customer requests. Compared to BL-random, FirstFit-profit is capable of maintaining higher utilization for most types of instances it rents. BL-small has higher utilization than FirstFit-profit; however, it fails to make profit due to that

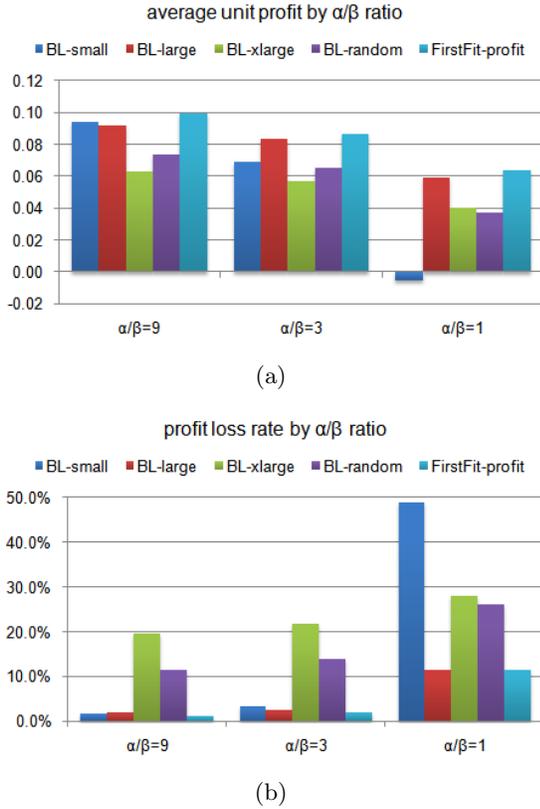
**Table 4: Instance utilization with regard to different instance types by  $\alpha/\beta$  ratio**

Algorithm	Instance type	$\alpha/\beta = 9$		$\alpha/\beta = 3$		$\alpha/\beta = 1$	
		number	utilization	number	utilization	number	utilization
BL-small	Small	106.2	92.2%	105.9	92.1%	106.0	92.3%
BL-large	Large	32.4	77.7%	32.1	77.9%	32.2	78%
BL-xlarge	Extra Large	22.3	63.4%	22.2	63.2%	22.1	63.7%
BL-random	Small	12.0	95.5%	12.0	95.4%	11.8	95.3%
	Large	12.2	77.0%	12.0	76.8%	12.1	77.4%
	Extra Large	12.1	59.1%	11.9	60.3%	11.9	59.6%
FirstFit-profit	Small	23.5	87.9%	1.6	48.4%	0	0%
	Large	16.5	92.3%	15.6	84.9%	13.1	83.3%
	Extra Large	2.7	65.9%	7.7	81.4%	9.7	81.5%

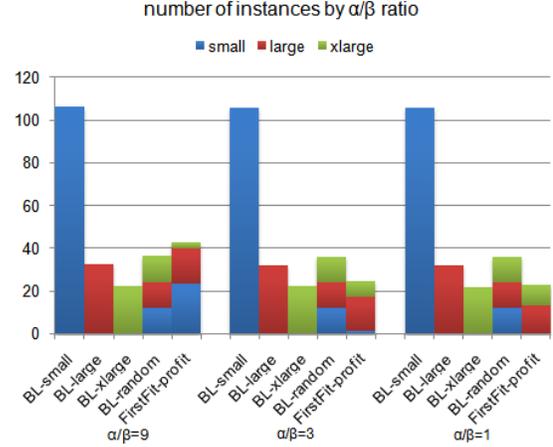
customer requests are not processed timely with the limited capacity of *small* instances.

On average, our FirstFit-profit algorithm achieves more profit than those baseline algorithms. In addition, FirstFit-profit uses less instances to process requests and delivers a higher instance utilization rate. Even though not our primary goal, improving the utilization of VM instances is a desirable effect for the underlying infrastructure provider as well because it can potentially improve the utilization of physical resources.

Note that, the FirstFit-satisfaction algorithm uses the same instance selection method as the FirstFit-profit algorithm, its instance number and utilization are similar to those of



**Figure 6: Average unit profit and profit loss rate by  $\alpha/\beta$  ratio (with satisfaction target 1.0). (a) average unit profit (\$ per time interval); (b) profit loss rate.**



**Figure 7: Number of instances with regard to different instance types by  $\alpha/\beta$  ratio.**

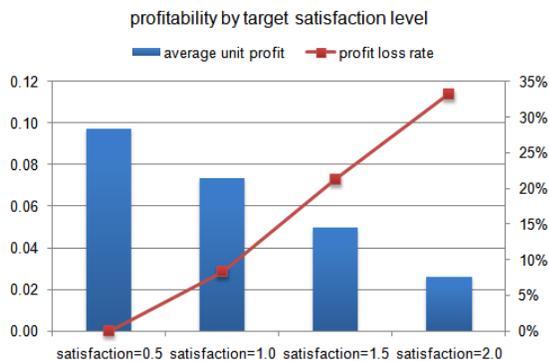
FirstFit-profit, hence we only provide the results of FirstFit-profit.

### 5.2.2 The impact of marginal rate of substitution $\alpha/\beta$

In Figure 6, Figure 7 and Table 4, we plot the results under different  $\alpha/\beta$  ratios. As shown in Figure 6, average unit profit declines and profit loss rate rises as  $\alpha/\beta$  reduces for all algorithms. A smaller  $\alpha/\beta$  means the customer prefers a shorter response time than a lower price, i.e., the service price should decline greatly as the response time rises slightly in order to keep the same level of satisfaction. In this situation, the service provider, with the aim of maximize profit (through charging the highest possible service price), should rent instances that can process the request with the minimum response time. From Figure 7 and Table 4, we can see that as  $\alpha/\beta$  reduces, our FirstFit-profit algorithm rents more *large* and *extra large* instances in exchange for shorter response time. It comes with the cost of higher instance prices, but offers the service provider to charge a higher service price, which eventually produces a higher profit. Baseline algorithms, which always bid for homogeneous instances, cannot dynamically handle different customer preferences.

### 5.2.3 The relationship between service profit and customer satisfaction

With information about customers' satisfaction levels and service performance, a service provider has the flexibility to



**Figure 8: Average unit profit (\$ per time interval) and profit loss rate achieved by the FirstFit-profit algorithm under different satisfaction targets.**

make tradeoffs between profit and customer satisfaction. In this section, we examine the relationship between the two.

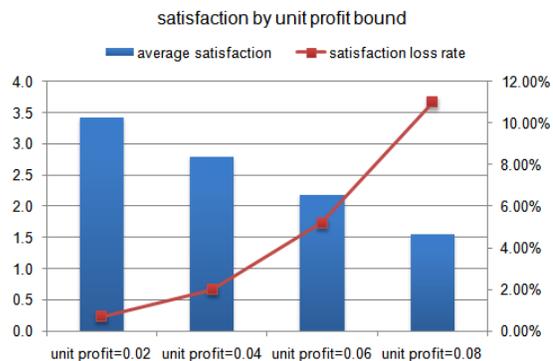
To simulate a set of customers with different preferences for service price and response time combination, we define a set of utility functions with  $\alpha/\beta$  set to 9, 3, 2, 1, 1/2, 1/4, 1/8 and an incoming customer request is randomly assigned a utility function from the set. We first observe the profitability of our FirstFit-profit algorithm under different satisfaction targets. Figure 8 shows the average unit profit and profit loss rate achieved by our FirstFit-profit algorithm under satisfaction levels at 0.5, 1.0, 1.5 and 2.0. It can be seen that average unit profit achieved by FirstFit-profit declines as the target satisfaction level rises. With the target satisfaction level increases from 0.5 to 2.0, average unit profit drops from 0.097 \$/time interval to 0.026 \$/time interval. In addition, the rise of target satisfaction level leads more requests to fail to make profit. The loss rate increases from 0% to 33%.

Similarly, we examine the impact of varying unit profit bound on the customer satisfaction. In Figure 9, we show satisfaction results achieved by our FirstFit-satisfaction algorithm under four unit profit bounds. As expected, it is clear that average satisfaction drops and satisfaction loss rate rises as unit profit bound increases. The average satisfaction drops by 54.8% in our FirstFit-satisfaction algorithm (from 3.43 to 1.55), while satisfaction loss rate increases from 1% to 11%.

From the results above, it can be concluded that service profit and customer satisfaction have a negative correlation. The service provider needs to pay the cost of profit reduction for the improvement of customer satisfaction, and vice versa. Furthermore, it also indicates that the service provider can optimize their profit by using flexible satisfaction levels, e.g., adopting a lower satisfaction level when workload is heavy or prices of VM instances are high and changing to a higher satisfaction level when workload is light or prices of VM instances are low. For service providers that want to improve customer satisfaction, it is also of considerable benefit to adopt a flexible unit profit bound.

## 6. RELATED WORK

Research on market-based resource allocation started from 1981 [25]. The tools offered by microeconomics for address-



**Figure 9: Average satisfaction and satisfaction loss rate achieved by the FirstFit-satisfaction algorithm under different unit profit bounds.**

ing decentralization, competition and pricing were thought useful in handling computing resource allocation problem [10]. Even though some market-based resource allocation methods are non-pricing-based [11, 14, 25], pricing-based methods can reveal the true needs of users who compete for shared resources and allocate resources more efficiently [15]. The application of market-based resource allocation ranges from computer networking [25], distributed file systems [14], distributed database [23] to computational job scheduling problems [8, 9, 24]. Our work is related to pricing-based computational job scheduling, or utility computing [19] as well as recent work on consistency rationing [13].

B.N. Chun et al. [7] built a prototype cluster that provided a market for time-shared CPU usage for various jobs. K. Coleman et al. [9] used market-based methods to address flash crowds and traffic spikes for clusters hosting Internet applications. Libra [21] was a scheduler built on proportional resource share clusters. One thing in common for [7, 9, 21] is that the value of a job does not change with the processing time. In [8], B. N. Chun et al. introduced time-varying resource valuation for jobs submitted to a cluster. The changing values were used to prioritize and schedule batch sequential and parallel jobs. The job with the highest value per CPU time unit was put ahead of the queue to run. D. E. Irwin et al. [12] extended the time-varying resource valuation function to take penalty into account when the value was not realized. The optimization therefore also minimized the loss due to penalty. In our model, the penalty is not directly reflected in our time-varying utility function, but implicitly reflected in the cost of physical resource usage in the cloud. The cost is incurred even when no revenue is generated.

F. I. Popovici and J. Wilkes [18] considered the scenario that a service provider rents resources at a price, which is similar to the scenario we deal with in cloud computing. The scheduling algorithm (FirstProfit) proposed in [18] used a priority queue to maximize the per-profit for each job independently. Our previous work [16] extended the time-varying valuation function to handle mashup services in the cloud.

Our algorithms differ from the above mentioned work mainly in the satisfaction model we develop. Our algorithms treat customer satisfaction as a variable according to utility the-

ory in economics while most existing algorithms treat it as a constant. This more realistic setting leaves more room for our algorithms to perform optimization. Moreover, our algorithms can handle different types of resources with different prices compared to many existing algorithms.

## 7. CONCLUSION

In this paper, we investigated the service provisioning problem at business service level in the cloud. Using utility theory leveraged from economics, we developed a utility model for measuring customer satisfaction. Based on the utility model, we gave a new type of SLAs between a business service provider and its customers. Our study revealed that the marginal rates of substitution of customers have a significant impact on a service provider's profit. We characterized the relationship between the profit of a service provider and customers' satisfaction based on the utility model. We proposed two scheduling algorithms for a service provider to make tradeoffs between its profit and customer satisfaction. By using flexible satisfaction targets (or unit profit bounds), our algorithms enable service providers to dynamically optimize their profit (or customer satisfaction) according to workload changes and resource price fluctuations. This work provided a practical method for supporting utility based SLAs in the cloud. Our extensive simulation based on Amazon EC2 data showed the effectiveness of the method.

## Acknowledgment

J. Chen is supported by a NICTA Research Project Award (NRPA). Professors Zomaya and Zhou are supported by an Australian Research Council Discovery Grant (DP1097111).

## 8. REFERENCES

- [1] Amazon EC2. <http://aws.amazon.com/ec2/>.
- [2] Amazon EC2 Spot Instances. <http://aws.amazon.com/ec2/spot-instances/>.
- [3] Amazon Web Services Case Studies. <http://aws.amazon.com/solutions/case-studies/>.
- [4] X264. <http://www.videolan.org/developers/x264.html>.
- [5] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. Above the clouds: a berkeley view of cloud computing. *Technical Report UCB/EECS-2009-28, EECS Department, University of California at Berkeley*, February 2009.
- [6] R. Brealey, S. Myers, and F. Allen. *Principles of Corporate Finance*. McGraw Hill, 2008.
- [7] B. N. Chun and D. Culler. Market-based proportional resource sharing for clusters. *Technical Report CSD-1092, University of California at Berkeley*, January 2000.
- [8] B. N. Chun and D. E. Culler. User-centric performance analysis of market-based cluster batch schedulers. In *Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID)*, pages 30 – 38, May 2002.
- [9] K. Coleman, J. Norris, G. Candea, and A. Fox. Oncall: defeating spikes with a free-market application cluster. In *Proceedings of the IEEE Conference on Autonomic Computing*, 2004.
- [10] D. F. Ferguson. *The application of microeconomics to the design of resource allocation and control algorithms*. PhD thesis, Columbia University. 1989.
- [11] D. F. Ferguson, C. Nikolaou, J. Sairamesh, and Y. Yemini. Economic models for allocating resources in computer systems. In S. H. Clearwater, editor, *Market-Based Control: A Paradigm for Distributed Resource Allocation*. World Scientific, 1996.
- [12] D. E. Irwin, L. E. Grit, and J. S. Chase. Balancing risk and reward in a market-based task service. In *Proceedings of 13th IEEE Symposium on High Performance Distributed Computing (HPDC)*, pages 160 – 169, June 2004.
- [13] T. Kraska, M. Hentschel, G. Alonso, and D. Kossmann. Consistency rationing in the cloud: Pay only when it matters. *PVLDB*, 2(1):253–264, 2009.
- [14] J. F. Kurose and R. Simha. A microeconomic approach to optimal resource allocation in distributed computer systems. *IEEE Trans. on Computers*, 38(5), 1989.
- [15] K. Lai. Markets are dead, long live markets. *Sigecom Exchanges*, 5(4):1 – 10, July 2005.
- [16] Y. C. Lee, C. Wang, A. Y. Zomaya, and B. B. Zhou. Profit-driven service request scheduling in clouds. In *Proceedings of the 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGRID)*, pages 15–24, 2010.
- [17] G. Mankiw. *Principles of economics*. Saurth-Western Pub, 2008.
- [18] F. I. Popovici and J. Wilkes. Profitable services in an uncertain world. In *Proceedings of the ACM/IEEE Supercomputing Conference (SC)*, November 2005.
- [19] M. Rappa. The utility business model and the future of computing services. *IBM Systems Journal*, 43(1):32 – 42, 2004.
- [20] J. Schad, J. Dittrich, and J.-A. Quiané-Ruiz. Runtime measurements in the cloud: Observing, analyzing, and reducing variance. *PVLDB*, 3(1):460–471, 2010.
- [21] J. Sherwani1, N. Ali, N. Lotia1, Z. Hayat, and R. Buyya. Libra: a computational economy-based job scheduling system for clusters. *Softw. Pract. Exper.*, 34:573 – 590, 2004.
- [22] M. Stokely, J. Winget, E. Keyes, C. Grimes, and B. Yolken. Using a market economy to provision compute resources across planet-wide clusters. In *Proceedings of the IEEE International Symposium on Parallel and Distributed Processing (IPDPS)*, pages 1–8, 2009.
- [23] M. Stonebraker, P. M. Aoki, W. Litwin, A. Pfeffer, A. Sah, J. Sidell, C. Staelin, and A. Yu. Mariposa: a wide-area distributed database system. *The VLDB Journal*, 5:48 – 63, 1996.
- [24] C. A. Waldspurger, T. Hogg, B. Huberman, J. O. Kephart, and W. Stornetta. Spawn: a distributed computational economy. *Software Engineering*, 18(2):103 – 117, 1992.
- [25] Y. Yemini. Selfish optimization in computer networks. In *Proceedings of the 20th IEEE International Conference on Decision and Control*, 1981.