2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing

SLA-based Resource Allocation for Software as a Service Provider (SaaS) in Cloud Computing Environments

Linlin Wu, Saurabh Kumar Garg and Rajkumar Buyya
Cloud Computing and Distributed Systems (CLOUDS) Laboratory
Department of Computer Science and Software Engineering
The University of Melbourne, Australia
{linwu,saurabh, raj}@csse.unimelb.edu.a

***Abstract*— Cloud computing has been considered as a solution for solving enterprise application distribution and configuration challenges in the traditional software sales model. Migrating from traditional software to Cloud enables on-going revenue for software providers. However, in order to deliver hosted services to customers, SaaS companies have to either maintain their own hardware or rent it from infrastructure providers. This requirement means that SaaS providers will incur extra costs. In order to minimize the cost of resources, it is also important to satisfy a minimum service level to customers. Therefore, this paper proposes resource allocation algorithms for SaaS providers who want to minimize infrastructure cost and SLA violations. Our proposed algorithms are designed in a way to ensure that Saas providers are able to manage the dynamic change of customers, mapping customer requests to infrastructure level parameters and handling heterogeneity of Virtual Machines. We take into account the customers' Quality of Service parameters such as response time, and infrastructure level parameters such as service initiation time. This paper also presents an extensive evaluation study to analyze and demonstrate that our proposed algorithms minimize the SaaS provider's cost and the number of SLA violations in a dynamic resource sharing Cloud environment.**

Keywords- Cloud computing; Service Level Agreement (SLA); Resource Allocation; Scheduling; Software as a Service.

I. INTRODUCTION

Traditionally the shrink-wrapped software sales model dominated the market. This model requires customers are required to purchase per petual or subscription-based license and manage the deployment themselves, including transitioning between different versions. Hence, customers need technical expertise and high initial investment for buying software. They also need to pay for upgrades as annual maintenance fee. With the emergence of Software as a Service (SaaS), applications are moving away from PC-based or ownership-based programs to web delivered hosted services [19]. The software services are provisioned on a pay-as-you-go basis to overcome the limitation of the traditional software sales model. Using the SaaS model, providers gain steady, on-going revenue from their customers. In exchange for the on-going charges, the customers get the benefit of continuously maintained software. Hence, there is no additional license fee for new versions and the complexity of transitioning to new releases is managed by SaaS providers. Due to the SaaS model's flexibility, scalability and cost-effectiveness, it has been increasingly adopted for distributing many enterprise software systems, such as banking, e-commerce business software [7][9]. SaaS providers such as Computer Associates (CA) [16] derive their profits from the margin between the operational cost of infrastructure and the revenue generated from customers. Therefore, SaaS providers are looking into solutions that minimize the overall infrastructure cost without adversely affecting the customers. Hence, the focus of this paper is on exploring policies to minimize the required infrastructure to meet customer demand in the context of SaaS providers offering hosted software services.

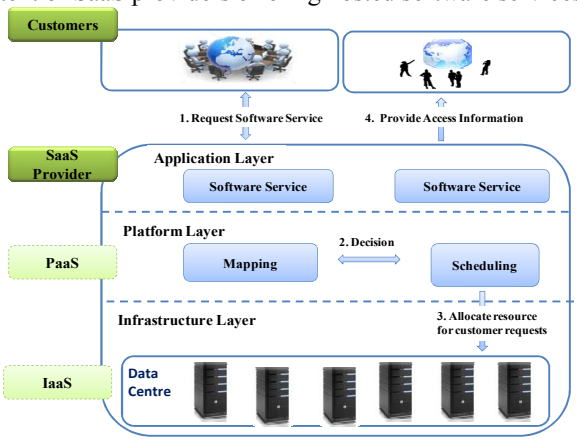


Figure 1. A system model of a SaaS layer structure

A SaaS model for serving customers in Cloud is shown in Fig. 1. A customer sends requests for utilizing enterprise software services offered by a SaaS provider, who uses three layers, namely application layer, platform layer and infrastructure layer, to satisfy the customer's request. The application layer manages all application services that are offered to customers by the SaaS provider. The platform layer includes mapping and scheduling policies for translating the customer's Quality of Service (QoS) requirements to infrastructure level parameters and allocating Virtual Machines (VMs) to serve their requests. The infrastructure layer controls the actual initiation and removal of VMs. The VMs can be leased from IaaS providers such as Amazon EC2 or private virtualized clusters owned by the SaaS provider. In both cases, the minimization of the number of VMs will deliver savings. The savings are greater when SaaS providers use the third party IaaS providers since no capital expenditure is required.

978-0-7695-4395-6/11 $26.00 © 2011 IEEE
DOI 10.1109/CCGrid.2011.51

Currently, SaaS providers such as Compiere ERP provide an individual VM for each customer [17] to maintain service level requirements in terms of response time and capacity. However, this causes wastage of hardware resources which results in high infrastructure cost since customers may not use complete VM capacity which is reserved to serve their requests. A multi-tenancy approach can reduce the needed infrastructure, but care must be taken in providing access to resources so that Service Level Agreements (SLAs) are not violated.

The current works in Cloud computing [13][10][2] are focused mostly on maximizing the profit of IaaS providers, but works related to the SaaS provider considering SLAs are still in their infancy. Many works do not consider the customer driven management, where resources have to be dynamically rearranged based on customers' demands. Thus, in this paper, we examine the resource allocation strategies, which allow a cost effective usage of resources in Clouds to satisfy dynamically changing customer demands in line with SLAs.

Therefore, in order to achieve the SaaS provider's objective to maximize profit and customer satisfaction levels, our work proposes cost effective mapping and scheduling policies which minimize cost by optimizing the resource allocation within a VM. These policies also take into account QoS parameters, and infrastructure heterogeneity regarding multiple types of VMs and various service initiation times. To satisfy customer's request in order to enlarge market share and minimize cost, the following questions have to be addressed:

- How to manage the dynamic change of customer requests? (such as upgrade from Professional to Enterprise product, add more user accounts for the same product)
- How to map customer requirements to infrastructure level parameters?
- How to deal with the infrastructure level heterogeneity?

The key contributions of this paper are as follows:

- It defines SLA with customers based on QoS parameters.
- It describes the mapping strategy by interpreting customer request requirements to infrastructure level parameters.
- It designs and implements scheduling mechanisms to maximize an SaaS provider's profit by reducing the infrastructure cost and minimizing SLA violations. The scheduling mechanism determines where and which type of VM has to be initiated by incorporating the heterogeneity of VMs in terms of their price, dynamic service initiation time, and data transfer time. In addition, it manages to reduce incurred penalties for handling dynamic service demands when customers are sharing resources.

This paper also presents a performance analysis of the proposed algorithms based on the customer's perspective: (i) arrival rate, (ii) proportion of upgrade requests; from SaaS providers' perspective: (i) service initiation time, (ii) penalty rate. The experimental results show that the proposed algorithms provide better solutions in terms of total profit and number of VMs when compared with base algorithm in most of the scenarios.

The rest of the paper is organized as follows. In Section II, we discuss prior works related to SLA-based and profit driven resource allocation in Cloud computing contexts. We also identify how our work differs from related works. Section III presents the detailed scenario and outlines the SLA supporting QoS parameters. Section IV describes a reference and two proposed algorithms, which are *ProfminVio*, *ProfminVmMaxAvaiSpace* and *ProfminVmMinAvaiSpace*. Section V firstly presents the experimental methodology including test bed and evaluation metrics; secondly, discusses the overall comparison of the performance evaluation results; thirdly, compares the algorithms by providing insights on when each algorithm should be used. Finally, Section VI concludes the paper by summarizing the comparison results and future work proposals.

II. RELATED WORK

Research on market driven resource allocation was started in early 80s [8][5]. Most of the market-based resource allocation methods are either non-pricing-based [10] or designed for fixed number of resources, such as FirstPrice [3] and FirstProfit [6]. Our work is related to user driven SLA-based profit maximization resource allocation for SaaS providers.

Reig G. et al [11] contributed on minimizing the resource consumption for serving requests and executing them before its deadline with a prediction system. Their prediction system enables the scheduling policies to discard the service of a request if the available resource capability is not able to complete the request before its deadline. However, in our work, we consider the enterprise applications which are different from compute and scientific applications.

Fu Y. et al [21] proposed an SLA-based dynamic scheduling algorithm (Squeeze) of distributed resources for streaming. Moreover, Yarmolenko V. et al [22] evaluated various SLA-based scheduling heuristics on parallel computing resources using resource (number of CPU nodes) utilization and income as evaluation metrics. Nevertheless, our work focuses on scheduling enterprise applications on VMs in Cloud computing environments. (The minimum unit of resources in our work is the number of VMs).

Popovici et al. [6] mainly considered QoS parameters on the resource provider's side such as price and offered load, but did not focus on the user side. However, our proposed work differs on QoS parameters from both the customer's and the SaaS provider's point of view and focuses on user driven scenarios.

Lee et al. [2] investigated the profit driven service request scheduling for workflow. In contrast, our work a) focuses on SLA driven QoS parameters on both user and provider sides, and b) solves the challenge of dynamic changing customer requests to gain profit and improve reputation.

In the context of the resource allocation algorithms for enterprise applications, Song et al. [18] presented the genetic algorithms in virtualized environments. However, the genetic algorithms generally require a long execution time. The long execution time increases the probability of SLA violation in the Cloud computing environments, where customers need to be served immediately.

In summary, this paper is unique in the following aspects:

- It manages the customer satisfaction level based on customer QoS requirements in minimizing the SLA violation and cost to increase the revenue, which is absent from most previous works in Cloud computing environments.
- The utility function is time-varying by considering dynamic VM deployment time (*aka* initiation time).
- It adapts to dynamic resource pools and consistently evaluates the cost of adding a new instance or removing instances, while most previous work deal with fixed size of the resource pools.

III. System Model

We consider the customers' requests for the enterprise software services from a SaaS provider by agreeing to the pre-defined SLA clauses and submitting their QoS parameters. Customers can dynamically change their requirements and usage of the hosted software services. The SaaS provider can use their own infrastructure or outsourced resources from public IaaS providers. For instance, "Saleforce.com" provides CRM software as a service using its own infrastructure, and "Force.com" offers this software using third party infrastructure [15]. The SaaS provider's objective is to schedule a request such that its profit is maximized while the customers' (QoS) requirements are assured. The platform layer of a SaaS provider uses mapping and scheduling mechanisms to interpret and analyze the customers' QoS parameters, and allocates respectively.

In this section, we explain the detailed system model from both the customers' and the SaaS providers' perspective and also describe the related mathematical models.

A. Actors

The actors involved in our system model are described below along with their objectives, activities and constraints.

1) SaaS Providers

SaaS providers lease enterprise software as hosted services to customers. They are interested in maximizing profit and ensuring QoS for customers to enhance their reputation in the marketplace. In our context, an example of the business process between a SaaS provider and a customer is where a service provider (*SaaS X*) offers CRM or ERP software packages, which are offered as three types of products (for example, Standards, Professional and Enterprise) and accounts (for example, Group, Team and Department). When a customer (*Company X*) submits its first time rent request with product type (*Standards*), account type (*Group*), and the required number of accounts (*m*), the provider will allocate resources to serve this customer. At anytime, *Company X* may require an upgrade in the service by adding more accounts or software editions. Customers can request an upgrade of services dynamically at any time in practice. Thus a SaaS provider has to handle these requests intelligently in line with the requirements as set out in the SLA.

From a SaaS provider's point of view, there is a legal contract-SLA with any customer and if any party violates SLA terms, the defaulter has to pay for the penalty according to the clauses defined in the SLA. The SLA properties include SaaS provider pre-defined parameters and the customer specified QoS parameters.

The properties defined in the SLA are as follows:

- **Request Type (*reqType*)**: It defines the customer request type, which is 'fist time rent' or 'upgrade service'. 'First time rent' means the customer is the customer who is renting a new service from this SaaS provider. 'Upgrade service' includes two types of upgrade, which are 'add account' and 'upgrade product'.
- **Product Type (*proType*)**: The software product offered to customers. For example, *SaaS X* offers Standard, Professional, and Enterprise product. The Standard product includes *Order and Sales* functions. The Professional contains all functions of Standard plus *Accounting* function. The Enterprise includes all features of Professional plus *Report* functions.
- **Account Type (*accType*)**: It constrains the maximum number of accounts a customer can create. For example, *SaaS X* offers three types of account: Group, Team, and Department, which allows each customer to create up to *m*, *2m* and *5m* number of accounts respectively.
- **Contract Length (*conLen*)**: How long the software service is legally available for a customer to use (minimum is one month).
- **Number of Accounts (*accNum*)**: The actual number of accounts that a customer wants to create. (Must be ≤ the maximum number of accounts for particular account type). For example, a customer *Company X* wants to rent Standard software and Group account type, then it can request [*1, m*] number of accounts.
- **Number of Records (*recNum*)**: The maximum number of records a customer is able to create for each account during the transaction and this will impact the data transfer time during the service upgrade. (The value of this parameter is predefined in SLA).
- **Response Time (*respTime*)**: It represents the elapsed time between the end of a demand on a software service and the beginning of a service. Violation occurs when actual elapsed time takes longer than pre-defined response time in SLA. We consider three types of response time: (i) response time for the first time renting of the service - *respTime(ftr)*, (ii) response time for adding new

accounts - *respTime(upSev,addAcc)* and (iii) response time for upgrading the product - *respTime(upSev,upPro)*. (The value of each type of response time is different and predefined in SLA).

The platform layer (Fig. 1) of a SaaS provider uses VM images to create instances according to the mapping decision. Therefore, it is important to identify the following properties for the resource allocation mechanisms to assure the SLA is adequately drafted:

- **VM types (*l*)**: How many types of VM can be used and what are they? For example, there are three types of VM, which are large, medium and small. The capacity of large VM equals to two medium VMs or four small VMs.
- **Service Initiation Time (*iniTimeSev*)**: How long it takes to initiate a VM, which is deployed with a type of software product.
- **VM Price (*PriVM*)**: How much it costs to a SaaS provider for using a VM to serve the customer request per time unit? It includes the physical equipment, power, network and administration price.
- **Data Transfer Time (*dataTrafT*)**: How long does it take to transfer a Gigabyte data from one VM to another?
- **Data Transfer Speed (*dataTrafSpeed*)**: It depends on the location distance and the network performance.

2) Customers

When a customer agrees with pre-defined SLA properties (such as response time), a request for an enterprise application is sent to the SaaS provider's application layer with the customer's QoS requirements (including **request type, product type, account type, contract length, and number of accounts)**.

B. Mapping Strategy: Mapping of customer QoS requirements to resources

The way of interpreting customer requirements to infrastructure level parameters depends on the resource capability. In our work, the infrastructure layer focuses on the VM level but not the host level. To be practical, we use the same record model as 'Saleforce.com' to restrict each account so as to create the maximum *N* number of records. An example of the mapping strategy between VM type, service product type, and the maximum and minimum number of accounts for each account type is described in Table I.

TABLE I. THE SUMMARY OF MAPPING BETWEEN RQUESTS AND RESOURCES

VM Type	Product Type	Account Type	Max Account #	Min Account #
Small	Standard	Group	m	1
Medium	Standard, Professional	Team.	2m	m+1
Large	Standard Professional, Enterprise,	Department	5m	2m+1

C. Mathematical Models

1) Profit Model

Let *C* be the number of customer requests and *c* indicates customer request id. Let at a given time instance *t*, a customer *c* submits a service request to the SaaS Provider. The customer specifies a product type, account type, contract length (*conLen*), number of accounts after agreeing with the pre-defined SLA clauses (response time). After the agreement, based on the SLA, the SaaS provider will reserve the requested software services which are translated at the infrastructure level as VM capacity.

Let *I* be the number of initiated VMs, and *i* indicates the VM id. Let *L* indicate the types of VMs, where for a particular VM_i with type *l* (VM_{il}) has $PriVM_{il}$ price. Let $iniTimeSev_{il}$ be the time taken for initiating service using VM_{il}.

Let $PriServ^c$ be the SaaS provider's final charge from customer *c* per month, which is subject on the product type, and account type. Let *P* indicates all product types. Let $Cost_{il}^c$ be the cost incurred to the SaaS provider by serving the customer *c* with VM_{il}. The $Prof_{il}^c$ indicates the profit for serving customer request *c* using VM_{il} . Then, the total profit $\sum_{c=1}^{C} \Pr of_{il}^c$ gained by the SaaS provider for serving total *C* number of customer requests is defined in Eq.(1)

$$\sum_{c=1}^{C} \Pr of_{il}^c = \sum_{c=1}^{C} \Pr iServ^c \times \sum_{c=1}^{C} conLen - \sum_{c=1}^{C} Cost_{il}^c$$

where, $\forall i \in I, l \in L, c \in C$ (1)

For a customer request *c*, the final service price $PriServ^c$ is subjected to the product type and account type. Let $Cost_{il}{}^c$ indicate the cost for serving request *c* with VM_{il} and it depends on the VM cost ($VMCost_{il}^c$) and penalty cost ($PenalytCost_{il}{}^c$.).

$$Cost_{il}^c = VMCost_{il}^c + PenaltyCost_{il}^c$$

where, $\forall i \in I, l \in L, c \in C$ (2)

The VM cost depends on the VM type *l*, the price of VM *i* with type *l* ($PriVM_{il}$), the service initiation Time ($iniTimeSev_{il}$) and service length (or contract length *conLen*) of customer request *c*. Eq. (3) defines the VM Cost.

$$VMCost_{il}^c = \Pr iVM_{il} \times \left(iniTimeSev_{il}^c \times conLen^c \right)$$

where, $\forall i \in I, l \in L, c \in C$ (3)

The SLA violation penalty (*Penalty*) model is similar to other related works [1][3][4] and is modeled as a Linear function. In Eq. (4), β is the penalty rate and *DT* is delay time.

$$Penalty = \alpha + \beta \times DT \quad (4)$$

The penalty function penalizes the service provider by reducing the utility (profit). According to the penalty model,

$$PenaltyCost_{il}^c = \alpha + \beta(reqType) \times delayTime_{il}^c(reqType)$$

where $\forall i \in I, l \in L, c \in C$ (5)

The penalty cost depends on the penalty delay time $delayTime_{il}^{c}$, penalty rate (β) and a constant number (α). The delay time and rate are subjected to the request type.
The delay time is the time variation between the response time specified in SLA and the actual time taken for customers to wait for the service response.

TABLE II. THE SUMMARY OF PENALTY DELAY TIME ACCORDING TO REQUEST TYPES

Table Head	First Time Rent	Upgrade Service	
		Add account	*Upgrade product*
SLA pre-defined response time	respTime (ftr)	respTime(upSev, addAcc)	respTime(upSev, upPro)
Actual Time	Service initiation time(iniTimeSev)	Service initiation time (iniTimeSev) Data transfer time(dataTrafT)	Service initiation time(iniTimeSev) Data transfer time(dataTrafT)

$$delayTime_{il}^{c} = \begin{cases} iniTimeSev - respTime(ftr) \\ \text{where, } reqType \text{ is first time rent} \quad (6) \\ iniTimeSev + \sum_{n=1}^{N} dataTrafT - respTime\ (upSev) \\ \text{where, } reqType \text{ is upgrade service} \quad (7) \end{cases}$$

There are three situations in which penalty delay can occur (Table II). If the request type is 'first time rent', the delay (violation) can occur due to long service initiation time. If the request type is 'upgrade service', the delay may be caused in adding accounts which may lead to service initiation time and data transfer time (if there is available initiated VMs), or caused by upgrade service product type which depends on service initiation and data transfer time for the total number of records created by previous request *c'* from the same company.

The service initiation time varies subjected to the physical machine's capability. The total data transfer time depends on the records data created by previous request *c'* of the same company. More specific, number of account created by previous request *c'*- ($accNum^{c'}$), number of records created per account ($recNum^{c'}$), the total record size ($\sum_{n=1}^{N} recSize^{c'}$) (GB) and data transfer time per GB ($dataTranfT$). *N* indicates the total number of records and *n* is the record id.

$$\sum_{n=1}^{N} dataTrafT = accNum^{c'} \times recNum^{c'} \times \sum_{n=1}^{N} recSize^{c'} \times dataTrafT$$
$$\text{where } \forall n \in N, c' \in C \quad (8)$$

IV. ALOGORITHM

The main objective of our work is to maximize the profit for a SaaS provider by minimizing the cost of VMs using effective platform layer resource allocation strategies. As noted earlier, the current SaaS providers such as Compiere ERP provide an individual VM for each customer [17] to maintain service level requirements in terms of response time and capacity. We implemented this scenario as a base algorithm by initiating a new VM for each individual company in order to minimize the SLA violation (*ProfminVio*). To optimize the profit further, we propose two SLA-based profit maximization algorithms.

A. Base Algorithm: Maximizing the profit by minimizing the number of SLA violations (ProfminVio)

A SaaS provider can minimize SLA violations by serving each individual company with a new VM involving the two main request types: a) first time rent and b) upgrade service. The algorithm first checks the request type, if the request type is 'first time rent' service then it finds the most suitable VM type *l* by using mapping strategy described in Table I, and afterwards calculates and records the profit for initiating a new VM using Eq. (1) and (2). Otherwise if the request type is 'upgrade', then check which type of upgrade is requesting. If upgrade type is 'add account', then the company's permissions are updated by allowing access to more users on VM_i which is serving this company. If the upgrade type is 'upgrade product', then first it finds the most suitable VM type *l* (Table I) and assigns request *c* to it, then calculates and record the profits.

This algorithm reduces the number of violations by using a new VM for each company to guarantee the response time. However, it is costly because a large number of VMs are initiated.

B. Proposed Algorithms

We propose the following two algorithms:

- Maximizing the profit by minimizing the cost by reusing VMs, which have maximum available space (*ProfminVmMaxAvaiSpace*).
- Maximizing the profit by minimizing the cost by reusing VMs, which have minimum available space (*ProfminVmMinAvaiSpace*).

1) ***Algorithm 1:*** *ProfminVmMaxAvaiSpace*

A SaaS provider can maximize the profit by minimizing the resource cost, which depends on the number and type of initiated VMs. Therefore, this algorithm is designed to minimize the number of VMs by utilizing already initiated VMs. *Algorithm 1* describes the *ProfminVmMaxAvaiSpace* algorithm, which involves two main request types: a) first time rent and b) upgrade service. Let the request of a customer *c* includes request type (*reqType*), product type (*proType*), account type (accType), number of accounts (accNum). The algorithm checks the request type, if the request type is 'first time rent' then it finds the VM_i with type *l* (VM_{il}) that matches to the service request parameters using mapping table similar to Table I. Then, it checks whether there is already initiated VM_{il} and deployed with same type of product as customer *c* requested. If there is an initiated VM_{il} where product *proType* has been deployed as *customer c* requested, then the algorithm checks whether this

VM_{il} has enough space to place the request of customer *c* according to his/her requested number of accounts (*accNum*) and the available space on this VM. If there are more than one VM_{il} with enough available space to place the request *c*, then the request *c* is assigned to the machine with maximum available space (Worst-fit manner) as illustrated in Fig. 2. (The gray space indicates unavailable space, x axis indicates the id of VM with same type and deployed with same type of product as customer *c* requested; y axis indicates number of accounts a VM can hold). If there is no initiated VM with type *l*, then check the next type of VM - $VM_{i(l+1)}$ which is deployed with the same software product type as request *c* specified, repeat step (2 to 13)

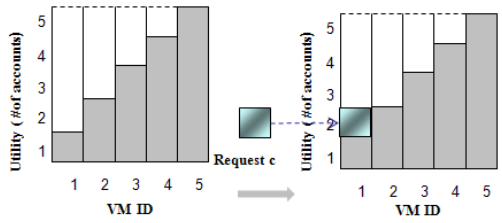


Figure 2. MaxAvaiSpace Strategy

If the request type is 'upgrade', then the type of upgrade is checked. If upgrade type is 'add account', then it first checks VM_i which has placed the previous request *c'* from the same company. If VM_i is not capable to place new request without exceeding the number of account limitation, then the suitable type *l* of VM is found that has the maximum available capacity to place request *c*. Then move the previous data from VM_i to new VM and release the space occupied by old request from VM_i. On the other hand, if upgrade to more advanced product edition, the new request is placed to the suitable VM by using the *MaxAvaiSpace* Strategy (Fig. 2) and then the customer data is migrated to new VM and release the space occupied by old request from VM_i.

Algorithm 1. Pseudo-code for *ProfminVmMaxAvaiSpace*
Input request (*c*) with QoS parameters , VM*i*
Output Boolean
Functions:
First Time Rent (*c*){
1 If (there is initiated VM_i with type *l* matches to the VM type requested by *c*) {
2 If (VM_i deployed the same product type as *c* required) {
3 For each initiated VM_i with type *l* (VM_{il}){
4 If (VM_i has enough space to place *c*)**{**
5 put VM_i into *vmList*
6 }
7 }
8 Sort(*vmList***)** according to the available space
9 Schedule to process *c* on ***VMmax***, which has maximum available space
10 }
11 Else {
12 Initiate new VM with type *l* and deploy the product type as request *c* required
13 }
14 }
15 Else While (*l+j*<=*L*) loop {
16 If (there is initiated VM with next type *l+j,* where type *l+j* matches to the VM type required by request *c*) {
17 Repeat from Step 2 to 13
18 j++
19 }
20 }
21}
Upgrade(*c*) {
1 If (upgrade type is 'add account'**) {**
2 get Id *i* and type *l* of VM, which processed the previous request from same company as *c*
3 If (VM_i has enough space to place *c*){
4 Schedule to process *c* on VM_i.
5 }
6 Else {
7 Repeat step 1 to 21 of **First Time Rent(*c*)**
8 Transfer data from old VM to new VM
9 Release space in old VM
10 }
11 }
12 If (upgrade type is **'upgrade service'){**
13 Repeat step 7 to 9 of **Upgrade(c)**
14 }
15}

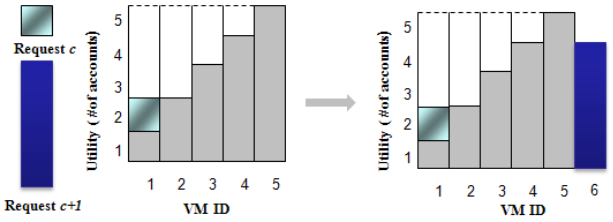


Figure 3. Example of disadvantage of *MaxBlankSpace* Strategy

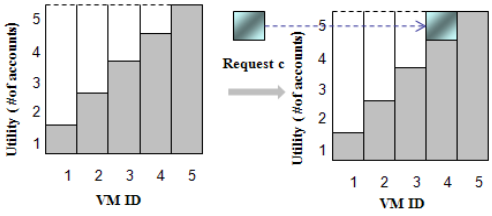


Figure 4. MaxAvaiSpace Strategy

This algorithm minimizes the number of initiated VMs in order to optimize profit. Moreover, it minimizes number of violations caused by service upgrade because the request *c* is scheduled to the VM, which has the maximum available space. In such a way, it reduces the penalty caused by upgrading service. However, the disadvantage of this algorithm is that it can decrease the profit in some cases; particularly when the maximum available space is occupied by small number of accounts and lead to requests (required large number of accounts) have to be served by a new VM. For example, in Fig. 3, VM 6 is a new VM initiated to serve request *c+1*, because VM 1 has been occupied by Request *c*.

*2) **Algorithm 2 :** ProfminVmMinAvaiSpace*

To overcome the disadvantages of algorithm 1, we reduce the space wastage by using minimum available space (*MinAvaiSpace*) Strategy (Fig. 4) instead of *MaxAvaiSpace* Strategy. When there are more than one *VM* with type *l*, deployed with the same product type as customer request *c* required, the VMs with enough available space to serve *c* are selected. Then request *c* is scheduled to the machine with the **minimum** available space (Best-fit manner) (Step 9). The rest of steps are the same as Algorithm 1.

V. PERFORMANCE EVALUATION

We present the performance results obtained from an extensive set of experiments. We have compared our algorithms with the scheduling strategy used by current SaaS providers such as Compiere, i.e., *ProfminVio.* In following sections, we first describe our experiment methodology, followed by performance metrics and detailed QoS parameters description. In subsequent sections, we present the analysis of the results showing the impact of the customer-side QoS parameters - (i) request arrival rate, (ii) proportion of upgrade request; and SaaS providers' side parameters - (i) service initiation time, (ii) penalty rate.

A. Experimental Methodology

CloudSim [11] is used to simulate the cloud computing environment that utilizes our proposed algorithms for resource allocation. We observe the performance of the proposed algorithms from both customers' and SaaS providers' perspectives. From customers' perspective, we observe how many SLAs are violated. From SaaS providers' perspective, we observe how much cost reduced and how many VMs are initiated. Therefore, there are three performance measurement metrics: total cost, number of initiated VMs, and percentage of SLA violations. All the parameters used in the simulation study are given in following sections.

1) Customers' Side

We examine our algorithms with 1000 customers. From customer side, two parameters (arrival rate and number of company upgrade) are varied to evaluate their impact on the performance of our proposed algorithms. Since there is no available workload specifying these parameters, we use normal distribution (standard deviation = (1/2)x mean) to model all parameters, except request arrival rate, which follows Poisson distribution.

- Five different types of request arrival rate are used by varying the mean from 200 to 650 customers per second.
- Five different sets of company request types are used by varying the mean from 20% to 80% of companies request upgrade service in order to vary the portion of service request type.

2) SaaS Providers' Side

The SaaS provider offers three types of software service products, and also three types of account types (Table 1). The cost per hour for using a VM (*PriVM*) in self-hosted VM follows the price schema of Amazon EC2 [14], because it is easier for SaaS provider to extend to public Cloud and actual cost of self-hosted VMs is less expensive than the price schema we are using. Resource price which are used for modeling VMs are shown in Table III.

- The five different types of average service initiation time are used in the experiment, and the mean service initiation time varied from 5 minutes to 15 minutes. The mean of initiation time is calculated by conducting real experiments of 60 samples on Amazon EC2 [14] done for four days (2 week days and 2 weekend days) by deployed different edition of products. The service initiation time is varied using normal distribution.
- The penalty cost (the same as in Eq. 5) is modeled by Eq. (5) (6) (7). It depends on the request type, product type and account type. The mean of Penalty Rate (β) is varied from value 2 (very low) to value 15 (very high).

TABLE III. THE SUMMARY OF VM PRICE

VM Type	Price($/hour)
Small	0.085
Medium	0.34
Large	0.68

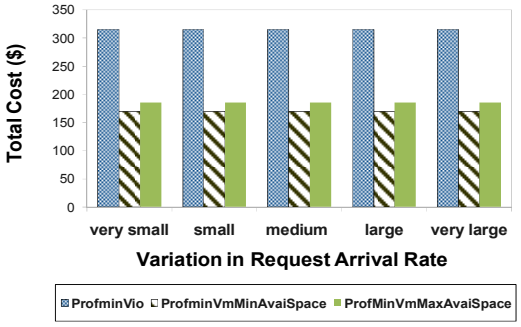
B. Impact of QoS parameters

We compare our proposed algorithms by examining the impact of QoS parameters on the performance metrics. All of results present the average obtained by 5 experiment runs. In each experiment, we vary one parameter, and the rest parameters are given constant mean value. The constant mean values are: arrival rate=1000 requests/sec, other parameters, and penalty rate factor (r) =10. In the following sections, we examine various experiments by varying both customer and SaaS provider side's SLA properties to analyze the impact of each parameter. The mean response time which is used to measure the violation equals 5 if the request type is 'first time rent', equals 10 if the request type is 'upgrade product' and 3 when the request type is 'add account.'

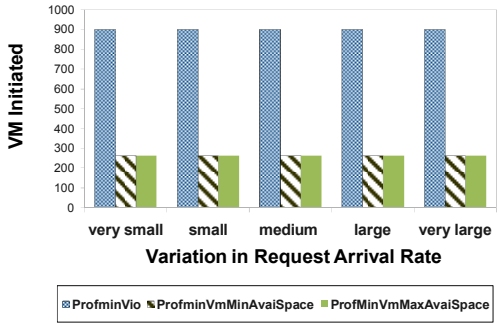
1) Impact of arrival rate variation

To observe the impact of arrival rate in our algorithms, we vary the arrival rate factor, while keeping all other factors as the same. All experiments are conducted with 1000 customers' requests. It can be seen from Fig. 5, in average the algorithm *ProfminVMminAvaiSpace* performs best to reduce about 50% cost by using approximately only 60% number of VMs compared with *ProfminVio*. As Fig. 5c shows, when the request arrival rate varies from 'large' to 'very large', the number of SLA violations caused by our proposed algorithms increases because when large number of concurrent requests comes, delaying the response time for upgrading services. Similar cost is generated by

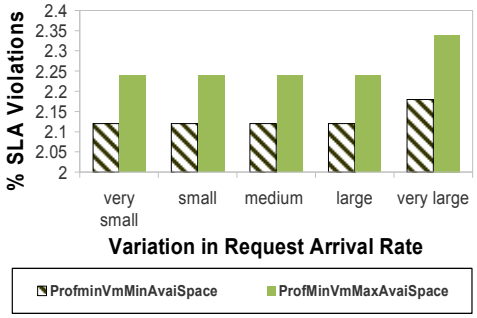
ProfminVMminAvaiSpace and *ProfminVMmaxAvaiSpace,* and the former one is slightly better than the latter one, because it costs less with similar number of VMs but generates less number of SLA violations. Therefore, during the variation of arrival rate, the *ProfminVMminAvaiSpace* perfoms best and optimizes the SLA violations in the context of resource sharing, where it is impossible to avoid SLA violations.



(a). Total Cost



(b). Number of initiated VMs



(c). Percentage of SLA Violations

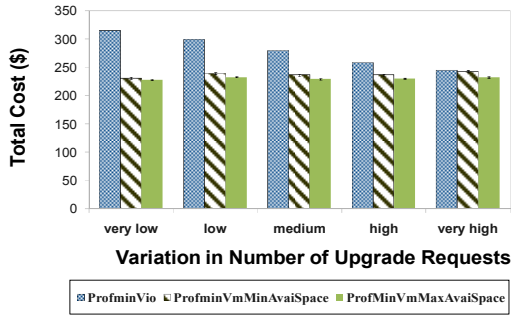
Figure 5. Impact of arrival rate variation

2) Impact of company upgrade frequency variation

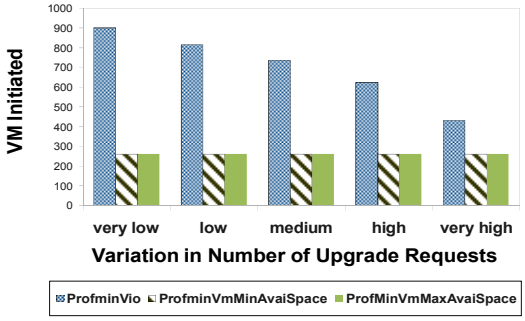
To investigate the impact of different proportion of request types ('first time rent', 'upgrade service') by varying the customer upgrade number. As it can be seen from Fig. 6, during the variation of number of customer upgrade, the total cost of *ProfminVio* decreases because it reduces number of VMs (reduced almost 49% on average) by utilizing the already initiated VMs for serving upgrades.

With given reduction in cost, our proposed algorithms cause very less number of violations as can be seen from the Fig. 6c, where the overall percentage of SLA violations is less than 13%, and only when the company upgrades percentage is 'very high', the percentage of SLA violations is higher than 8%. When the variation in company upgrade frequency is low (17% upgrade service requests), *ProfminVMmaxAvaiSpace* causes more SLA violations due to the disadvantage of MaxAvaiSpace Strategy. In the contrary, when the company upgrade frequency is very high, *ProfminVMminAvaiSpace* violates more requests because space has been occupied by 'first time rent' request types, leading to long upgrade time and even penalty delay.

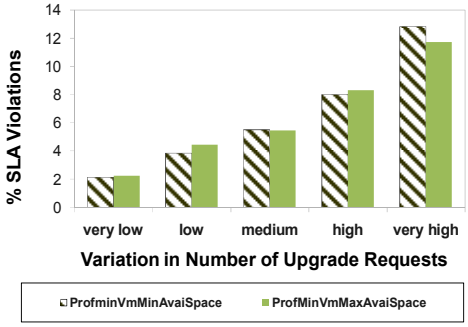
Therefore, when large numbers of requests are 'first time rent', the algorithm *ProfminVMminAvaiSpace* and *ProfminVMminAvaiSpace* cost less with less number of VMs, and with a bit more SLA violations compare with *ProfminVio* (no violation).



(a). Total Cost



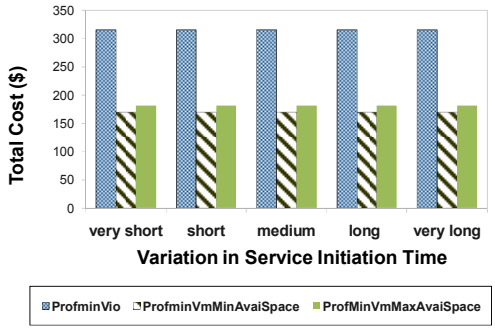
(b). Number of Initiated VMs



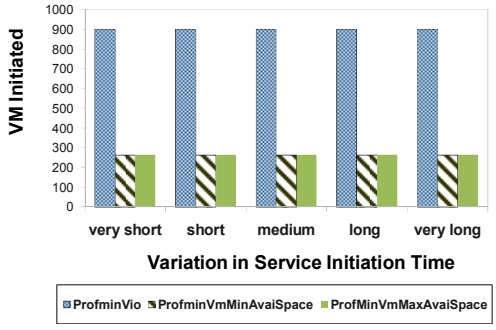
(c). Percentage of SLA Violations

Figure 6. Impact of number of upgrade requests

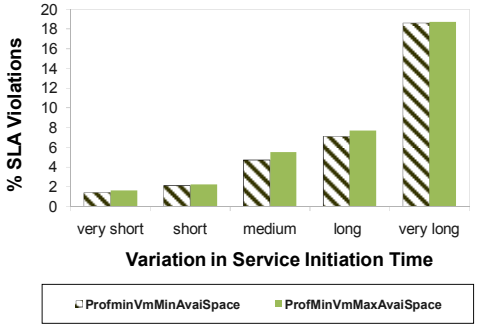
3) Impact of initiation time variation



(a). Total Cost



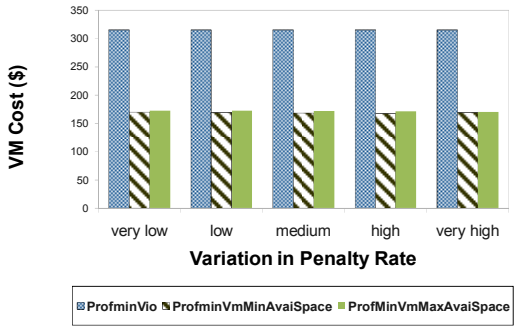
(b). Number of initiated VMs



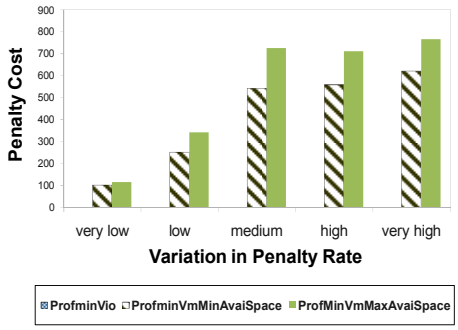
(c). Percentatge of SLA Violations

Figure 7. Impact of initiation time variation

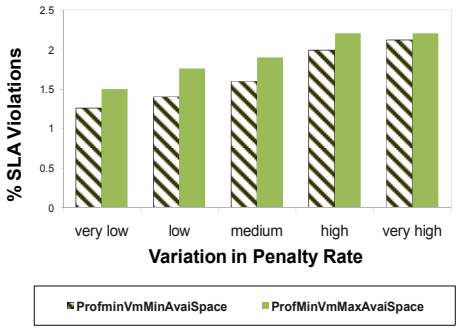
4) Impact of penalty rate variation



(a). VM Cost



(b). Penalty Cost



(c). Percentage of SLA Violations

Figure 8. Impact of penalty rate variation

The service initiation time also includes the VM initiation time for deploying software service as requested. Fig. 7 shows how the variation in service initiation time after accepting a company request on the SaaS provider's cost. When the initiation time varies from 'very short' to 'very long', the total cost of all algorithms increased slightly. The *ProfminVio* is impacted more because it initiated more VMs. The average SLA violation percentage of our algorithms is less than 13% When the initiation time is 'very short' and 'very long', the algorithm *ProfminVMminAvaiSpace* and *ProfminVMminAvaiSpace* generate a similar number of violations, because initiation time delay is the same for both of the algorithms.

We investigate how penalty rate (β) impacts our algorithms. It can be observed from Fig. 8 that *ProfminVmMinAvaiSpace* and *ProfminVmMaxAvaiSpace* are impacted when varying the penalty rate factor because they schedule customer requests with shared resources. The penalty cost of algorithms increases during variation of the penalty rate due to SLA violation increases because of resource sharing between multiple customers. However, when the SLA violation is very low the maximum percentage is less than 2.5%. In conclusion, Fig. 8 a and b show that our algorithms minimize the cost although penalty cost is increasing during penalty rate variation.

VI. CONCLUSION AND FUTURE WORK

In Cloud computing environments, primarily three types of on-demand services are available for customers i.e. Software as a Service, Infrastructure as a Service and

Platform as a Service. This paper focused on scheduling customer requests for SaaS providers with the explicit aim of cost minimization with dynamic demands handling. To achieve this goal, we answered questions raised in the introduction section by using mapping and scheduling mechanisms to deal with the customer side dynamic demands and resource level heterogeneity. Thus, we implemented three cost driven algorithms which considered various QoS parameters (such as arrival rate, service initiation time and penalty rate) from both the customers' and the SaaS providers' perspective. Simulation results show that on average, the *ProfminVMminAvaiSpace* algorithm optimized cost savings better when compared to the other proposed algorithms.

In building on the research undertaken in this paper in the future, we will analyze ways to increase the efficiency of the algorithms in terms of total profit and shall also consider the SLA negotiation process in Cloud computing environments to improve customer satisfaction levels. We would also like to add different types of services and other pricing strategies such as spot pricing to increase the profit for service providers. Moreover, investigating the knowledge based scheduling for maximizing a SaaS provider's profit to improve our algorithms' time complexity. Moreover, we will look into the penalty limitation by considering system failures.

ACKNOWLEDGMENT

We thank the anonymous reviewers, colleagues in Cloud Lab at the University of Melbourne and Mr. Bevan Mailman who helped us to improve the quality of this paper.

REFERENCES

[1] C.S. Yeo, and R. Buyya, "Service level agreement based allocation of cluster resources: Handling penalty to enhance utility". In Proceedings of the 7th IEEE International Conference on Cluster Computing (Cluster 2005), Bostan, MA, USA.

[2] Y.C. Lee, C. Wang, A.Y. Zomaya and B.B. Zhou, "Profit-driven Service Request Scheduling in Clouds". In Proceedings of the International Symposium on Cluster and Grid Computing, (CCGrid 2010), Melbourne, Australia.

[3] O. F. Rana, M. Warnier, T. B. Quillinan, F. Brazier, and D. Cojocarasu, "Managing Violations in Service level agreements". In proceedings of the 5th International Workshop on Grid Economics and Business Models (GenCon 2008), Gran Canaris, Spain.

[4] D.E. Irwin, and L.E. Grit, and J.S. Chase, "Balancing Risk and Reward in a Market-based Task Service". In Proceedings of the 13th International Symposium on High Performance Distributed Computing (HPDC 2004), Honolulu, HI, USA.

[5] Y. Yemini, "Selfish optimization in computer networks processing". In Proceeding of the 20th IEEE Conference on Decision and Control including the Symposium on Adaptive Processes, San Diego, USA.

[6] I. Popovici, and J. Wiles, "Proitable services in an uncertain world". In Proceeding of the18th Conference on Supercomputing (SC 2005), Seattle, WA.

[7] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud Computing and Emerging IT Platforms: Vision, Hype, and Reality for Delivering Computing as the 5th Utility, Future Generation Computer Systems", 25(6), (pp. 599-616), Elsevier Science, Amsterdam, The Netherlands.

[8] D. Parkhill, "The challenge of the computer utility", 1966, Addision-Wesley Educational Publishers Inc., USA.

[9] M. A. Vouk, "Cloud Computing-Issues, Research and Implementation". In Proceedings of 30th International Conference on Information Technology Interfaces (ITI 2008), Dubrovnik, Croatia.

[10] J. Broberg, S. Venugopal, and R Buyya, Market-oriented Grids and Utility Computing: The state-of-the-art and future directions, Journal of Grid Computing, 3(6), (pp.255-276).

[11] Rodrigo N. Calheiros, Rajiv Ranjan, Anton Beloglazov, Cesar A. F. De Rose, and Rajkumar Buyya, CloudSim: A Toolkit for Modeling and Simulation of Cloud Computing Environments and Evaluation of Resource Provisioning Algorithms, Software: Practice and Experience (SPE), Volume 41, Number 1, Pages: 23-50, ISSN: 0038-0644, Wiley Press, New York, USA, January, 2011.

[12] G Reig, J. Alonso,and J. Guitart, "Deadline Contrained Prediction of Job Resource Requirments to Manage High-Level SLAs for SaaS Cloud Providers". Tech. Rep. UPC-DAC-RR, Dept. d'Arquitectura de Computadors, University Polit'ecnica de Catalunya, Barcelona, Spain.

[13] S. K. Garg, R. Buyya, and H. J. Siegel, "Time and Cost Trade-off Management for Scheduling Parallel Applications on Utility Grids", Future Generation Computer Systems,26(8), (pp. 1344-1355).

[14] C. Vecchiola, X.C. Chu, M. Mattess, and R. Buyya, "Aneka—Integration of Private and Public Clouds", Cloud Computing Principles and Paradigms, Willy, USA, 2011

[15] Saleforce.com, Retrived on 6th Dec 2010: http://www.salesforce.com

[16] Computer Associates Pty Ltd. Retrived on 6th Dec 2010: http://www.ca.com

[17] Compiere ERP on Cloud, Retrieved on 6th Dec 2010: http:// http://www.compiere.com/

[18] Y. Song, Y. Li, H. Wang, Y. Zhang, B. Feng, H. Zang, Y. Sun, "A Service-Oriented Priority-Based Resource Scheduling Scheme for Virtualized Utility Computing", High Performance Computing-HiPC 2008.

[19] T. Gad, "Why Traditional Enterprise Software Sales Fail". July 2010, Retrieved on 6th Dec 2010: http://www.sandhill.com/opinion/editorial_print.php?id=307

[20] Y. Fu, A. Vahdat, "SLA Based Distributed Resource Allocation for Streaming Hosting Systems", http://issg.cs.duke.edu

[21] V. Yarmolenko and R. Sakellariou. " An Evaluation of Heuristics for SLA Based Parallel Job Scheduling". In Proceddings of the 3rd High Performance Grid Computing Workshop (in conjunction with IPDPS 2006). Rhodes, Greece.