The International Journal Of Engineering And Science (IJES)
||Volume||2 ||Issue|| 5 ||Pages|| 16-24||2013||
ISSN(e): 2319 – 1813 ISSN(p): 2319 – 1805

# Various Scheduling Algorithms for Resource Allocation In Cloud Computing

Swarupa Irugurala[1], Dr.K.Shahu Chatrapati[2]
[1]*Computer Science and Engineering, JNTUH, Kukatpally, Hyderabad, India*

**----------Abstract----------**

*To provide services to customers SaaS providers utilize resources of internal data centers or rent resources from a public Infrastructure as a Service (IaaS) provider. In-house hosting can increase administration and maintenance costs whereas renting from an IaaS provider can impact the service quality due to its variable performance. To overcome these limitations, we propose innovative admission control and scheduling algorithms for SaaS providers to effectively utilize public Cloud resources to maximize profit by minimizing cost and improving customer satisfaction level. Furthermore, we conduct an extensive evaluation study to analyze which solution suits best in which scenario to maximize SaaS provider's profit. Simulation results show that our proposed algorithms provide substantial improvement over reference ones across all ranges of variation in QoS parameters.*

***Keywords:*** *resource control, cloud computing, scheduling algorithm*

Date Of Submission: 18 April 2013 Date Of Publication: 13,May.2013

## I. INTRODUCTION

Cloud computing has emerged as a new paradigm for delivery of applications, platforms, or computing resources (processing power/bandwidth/storage) to customers in a "pay-as-you-go-model". The Cloud model is cost-effective because customers pay for their actual usage without upfront costs, and scalable because it can be used more or less depending on the customers' needs. Due to its advantages, Cloud has been increasingly adopted in many areas, such as banking, e-commerce, retail industry, and academy. Considering the best known Cloud service providers, such as Sale-force.com [3], Microsoft , and Amazon , Cloud services can be categorized as: application (Software as a Service – SaaS), platform (Platform as a Service – PaaS) and hardware resource (Infrastructure as a Service – IaaS).In this paper, we focus on the SaaS layer, which allows customers to access applications over the Internet without software related cost and effort (such as software licensing and upgrade).

The general objective of SaaS providers is to minimize cost and maximize customer satisfaction level (CSL). The cost includes the infrastructure cost, administration operation cost and penalty cost caused by SLA violations. CSL depends on to what degree SLA is satisfied. In general, SaaS providers utilize internal resources of its data centres or rent resources from a specific IaaS provider. For example, Saleforce.com hosts resources but Animoto rents resources from Amazon EC2. In-house hosting can generate administration and maintenance cost while renting resources from a single IaaS provider can impact the service quality offered to SaaS customers due to the variable performance .To overcome the above limitations, multiple IaaS providers and admission control are considered in this paper.

The rest of this paper is organized as follows. In Section 2, we present system and mathematical models. As part of the system model, we design two layers of SLAs, one between users and SaaS providers and another between SaaS and IaaS providers[9]. In Section 3, we propose three admission control and scheduling algorithms. In Section 4, we show the effectiveness of the proposed algorithms in meeting SLA objectives and the algorithms' capacity in meeting SLAs with users even in the presence of SLA violations from IaaS providers[6]. Simulation results show that proposed algorithms improve the profit compared to reference algorithms by varying all range of QoS parameters. Finally, in Section 5, we conclude the paper by summarizing the comparison results and future work.

## II. SYSTEM MODEL

In this section, we introduce a model of SaaS provider, which consists of actors and 'admission control and scheduling' system. The actors are users, SaaS providers, and IaaS providers. The system consists of application layer and platform layer functions. Users request the software from a SaaS provider by submitting their QoS requirements. The platform layer uses admission control to interpret and analyse the user's QoS parameters and decides whether to accept or reject the request based on the capability, availability and price of VMs[6]. Then, the scheduling component is responsible for allocating resources based on admission control decision. Furthermore, in this section we design two SLA layers with both users and resource providers, which are SLA(U) and SLA(R) respectively.

### 2.1. Actors

The participating actors involved in the process are discussed below along with their objectives and constraints.

### 2.2 .User

On users' side, a request for application is sent to a SaaS provider's application layer with QoS constraints, such as, deadline, budget and penalty rate. Then, the platform layer utilizes the 'admission control and scheduling' algorithms to admit or reject this request. If the request can be accepted, a formal agreement (SLA) is signed between both parties to guarantee the QoS requirements such as response time.

### 2.3. SaaS provider

A SaaS provider rents resources from IaaS providers and leases software as services to users. SaaS providers aim at minimizing their operational cost by using resources from IaaS providers, and improving Customer Satisfaction Level (CSL) by satisfying SLAs, which are used to guarantee QoS requirements of accepted users[10].

### 2.4. IaaS provider

An IaaS provider offers VMs to SaaS providers and is responsible for dispatching VM images to run on their physical resources. The platform layer of SaaS provider uses VM images to create instances. It is important to establish SLA with a resource provider – SLA(R), because it enforces the resource provider to guarantee service quality.

### 2.2 Profit model

In this section we describe mathematical equations used in our work. Let at a given time instant t, I be the number of initiated VMs, and *J* be the total number of IaaS providers. Let IaaS provider *j* provide $N_j$ types of VM, where each VM type *l* has $P_{jl}$ price. The prices/GB charged for data transfer-in and -out by the IaaS provider *j* are *in*Pri and *out*Pri $_j$ respectively. Let ($iniT_{ijl}$) be the time taken for initiating VM *i* of type *l*Let a *new* user submit a service request at submission time $subT^{new}$ to the SaaS provider. The *new* user offers a maximum price $B^{new}$ (Budget) to SaaS provider with deadline $DL^{new}$ and *Penalty Rate* $\beta^{new}$. Let $inDS^{new}$ and $outDS^{new}$ be the data-in and –out required to process the user requests.Let $Cost^{new}_{ijl}$ be the total cost incurred to the SaaS provider by processing the user request on VM i of type l and resource provider j. Then, the profit $Prof^{new}_{ij}$ gained by the SaaS provider is defined as

$$Prof^{new}_{ijl} = B^{new} - Cost^{new}_{ijl};\ \forall i \in I,\ j \in J,\ l \in N_j \qquad (1)$$

The total cost incurred to SaaS provider for accepting the new request consists of request's processing cost ($PC^{new}_{ijl}$), data transfer cost ($DTC^{new}_{jl}$), VM initiation cost ($IC^{new}_{ijl}$), and penalty delay cost ($PDC^{new}_{ijl}\ T$) (to compensate for miss deadline). Thus, the total cost is given by processing the request on VM *i* of type *l* on IaaS provider *j*.

$$Cost^{new}_{ijl} = PC^{new}_{ijl} + DTC^{new}_{jl} + IC^{new}_{ijl} + PDC^{new}_{ij};\ \forall i \in I,\ j \in J,\ l \in N_j \qquad (2)$$

The processing cost ($PC^{new}_{ijl}$) for serving the request is dependent on the new request's processing time ($procT_{ijl}^{new}$) an hourly price of $VM_{il}$ (type *l*) offered by IaaS provider *j*. Thus, $PC^{new}_{ij}$ is given by:

$$PC^{new}_{ijl} = procT_{ijl}^{new} \times P_{jl}, \quad \forall i \in I, \ j \in J, \ l \in N \qquad (3)$$

Data transfer cost as described in Eq. (4) includes cost for both data-in and data-out.

$$DTC^{new}_{jl} = inDS^{new} \times inPri_{jl} + outDS^{new} \times outPri_{jl}; \quad \forall j \in J, \ l \in N_j \qquad (4)$$

The initiation cost ($IC^{new}_{ij}$) of VM $i$ (type $l$) is dependent on the type of VM initiated in the data center of IaaS provider $j$

$$IC^{new}_{ijl} = = iniT_{ij} \times P_{jl}, \ \forall i \in I, \ j \in J, \ l \in N_j \qquad (5)$$

In Eq. (7), penalty delay cost ($PDC^{new}_{ij}$) is how much the service provider has to give discount to users for SLA(U) violation. It is dependent on the penalty rate ($\beta^{new}$) and penalty delay time ($PDT^{new}_{ij}$) period. We model the SLA violation penalty as linear function which is similar to other related works .

$$PDC^{new}_{ijl} = \beta^{new} \times PDT^{new}_{ijl}; \quad \forall i \in I, \ j \in J, \ l \in N_j \qquad (6)$$

To process any new request, SaaS provider either can allocate a new VM or schedule the request on an already initiated VM. If service provider schedules the new request on an already initiated $VM_i$ , the new request has to wait until VM $I$ becomes available. The time for which the new request has to wait until it start processing on VM $i$ is $^{3}{}^{K=1}\ procT^{k}$ , where $^{k}ijl\ K$ is the number of request yet to be processed before the new request. Thus, $PDT_{ijl}^{new}$ is given by:

$$PDT^{new}_{ijl} = t + \text{procT}^{k} + \text{procT}^{\text{new}} - \text{DL}^{\text{new,}} \text{ if new VM is not initiated} \qquad (7)$$

$DTT^{new}_{ijl}$ is the data transfer time which is the summation of time taken to upload the input ($inDT^{new}_{ill}$) and download the output data ($outDT^{new}_{ijl}$) from the $VM_{il}$ on IaaS provider $j$. The data transfer time is given by:

$$DTT^{new}_{ijl} = inDT^{new}_{ijl} + outDT^{new}_{ijl}; \quad \forall i \in I, \ j \in J, \ l \in N_j \qquad (8)$$

Thus, the response time ($T_{ijl}^{new}$) for the new request to be processed on $VM_{il}$ of IaaS provider j is calculated

Eq (9) and consists of VM initiation time ($iniT_{ijl}^{new}$), request's service processing time ($procT_{ijl}^{new}$), data transfer time *($DTT^{new}_{ijl}$),* and penalty delay time *($PDT^{new}_{ijl}$).*

$$T_{ijl}^{new} = procT_{ijl}^{new} + iniT_{ijl} + DTT^{new}_{ijl} \qquad (9)$$

The investment return *($ret^{new}_{ij}$)* to accept new user request per hour on a particular $VM_{il}$ in IaaS provider j is ncalculated based on the profit *($prof^{new}_{ijl}$)* and time *($T_{ijl}^{new}$)*:

$$ret^{new}_{ijl} = prof^{new}_{ijl} / (T_{ijl}^{new}); \forall i \in I, \ j \in J, \ l \in N_j \qquad (10)$$

## III. ALGORITHMS AND STRATEGIES

In this section, we present four strategies to analyze whether a new request can be accepted or not based on the QoS requirements and resource capability[11]. Then, we propose three algorithms utilizing these strategies to allocate resources. In each algorithm, the admission control uses different strategies to decide which user requests to accept in order to cause minimal performance impact, avoiding SLA penalties that decrease SaaS provider's profit[9]. The scheduling part of the algorithms determines where and which type of VM will be used by incorporating the heterogeneity of IaaS providers in terms of their price, service initiation time, and data transfer time.

**3.1. Strategies**

In this section, we describe four strategies for request acceptance: a) initiate new VM, b) queue up the new user request at the end of scheduling queue of a VM, c) insert (prioritize) the new user request at the proper position before the accepted user requests and, d) delay the new user request to wait all accepted users to finish.

**3.1.1. Initiate new VM strategy**

"Initiate new VM strategy", which first checks for each type of VMs in each resource provider in order to determine whether the deadline of new request is long enough comparing to the estimated finish time. The estimated finish time depends on the estimated start time, request processing time, and VM initiation time.If the new request can be completed within the deadline, the investment return is calculated (Eq. (10)). If there is value added according to the investment return, and then all related information (such as resource provider ID, VM ID, start time and estimated finish time) are stored into the potential schedule list.

**3.1.2. Wait strategy**

It verifies each VM in each resource provider if the flexible time of the new request is enough to wait all accepted requests in $vm_{il}$ to complete.

**3.1.3. Insert strategy**

"Insert strategy", which first checks verifies if any accepted request $u_k$ according to latest start time in $vm_{il}$ can wait the new request to finish. If the flexible time of accepted request ( $f\ T_{ijl}^{\ k}$ ) is enough to wait for a new user request to complete then the new request is inserted before request *k*.

**3.1.4. Penalty delay strategy**

Fig. 5 describes the flow chart of "*penalty delay strategy*", which first checks if the new user request's budget is enough to wait for all accepted user requests in $vm_i$ to complete after its deadline. Eq. (1) is used to check whether budget is enough to compensate the penalty delay loss, and then the investment return is calculated and the remaining steps are the same as those in initiate new VM strategy. This strategy is presented as function *canPenaltyDelay()* in algorithms.

**3.2. Proposed algorithms**

service provider can maximize the profit by reducing the infrastructure cost, which depends on the number and type of initiated VMs in IaaS providers' data centre. Therefore, our algorithms are designed in a way to minimize the number of VMs by maximizing the utilization of already initiated VMs. In this section, based on above strategies we propose three algorithms, which are *ProfminVM*, *ProfRS*, and *ProfPD*: In admission control phase, the algorithm analyses if the new request can be accepted either by queuing it up in an already initiated VM or by initiating a new VM[12]. Hence, firstly, it checks if the new request can be queued up by waiting for all accepted requests on any initiated VM – using *Wait Strategy* . If this request cannot wait in any initiated VM, then the algorithm checks if it can be accepted by initiating a new VM provided by any IaaS provider – using *Initiate New VM Strategy* . If a SaaS provider does not make any profit by utilizing already initiated VMs nor by initiating a new VM to accept the request, then the algorithm *rejects* the request . Otherwise, the algorithm gets the maximum investment return from all of the possible solutions . The decision also depends on the minimum expected investment return *(*$expInvRet^{new}_{ijl}$*)* of the SaaS provider. If the investment return $ret^{new}_{ijl}$ is more than the SaaS provider's $expInvRet^{new}_{ijl}$, the algorithm *accepts* the new request otherwise it *rejects* the request .

The **scheduling** phase is the actual resource allocation and scheduling based on the admission control result; if the algorithm *accepts* the new request, the algorithm first finds out in which IaaS provider $rp_j$ and which VM $vm_i$ a SaaS provider can gain the maximum investment return by extracting information from *PotentialScheduleList* . If the maximum investment return is gained by initiating a new VM , then the algorithm initiates a new VM in the referred resource provider ($rp_j$ ), and schedule the request to it. Finally, the algorithm schedules the new request on the referred VM ($vm_i$ ) . The time complexity of this algorithm is *O (R J + R),* where *R* indicates the total number of requests and *J* indicates the number of resource providers.

**Algorithm 1.** Pseudo-code for *ProfminVM*
algorithm. **Input:** New user's request parameters
($u^{new}$), $expInvRet^{new}_{ij}$
**Output:** Boolean
**Functions:**

***admissionControl*()** {
**1.** **If** (there is any initiated VM) {
**For each** $vm_i$ in each resource provider
**2.** $rp_j$ {
**3.** **If (!** ***canWait***$(u^{new}, vm_i))$ {
**4.** **continue;**
**5.** }
**6.** }
**7.** }
**8.** **Else If (!** ***canInitiateNew***$(u^{new}, rp_j))$
**Return**
**9.** ***reject***
**If** (PotentialScheduleList is
**10.** empty)
**Return**
**11.** ***reject***
**12.** **Else** {
**13.** Get the $\boldsymbol{max}_{new}[ret^{new}_{ij}, SD_{ij}]$ in PotentialScheduleList
**14.** **If** $(\boldsymbol{max}(ret_{ij}) \geq expInvRet_{ij})$
**15.** **Return *accept***
**16.** **Else**
**17.** **Return *reject***
**18.** }
**19.**

}

***schedule*()** {
**20.** Get the $[ret^{new}_{max}, SD_{max}]$ in *maxRet(*PotentialScheduleList*)*
**21.** **If** ($SD_{max}$ is initiateNewVM)
**22.** initiateNewVM in $rp_j$
**23.** Schedule the $u^{new}$ in $VM_{max}$ in $rp_{max}$ according to $SD_{max}$.
}

### 3.2.1. Maximizing the profit by rescheduling (ProfRS)

In *ProfminVM* algorithm, a new user request does not get priority over any accepted request. This inflexibility affects the profit of a SaaS provider since many urgent and high budget requests will be rejected. Thus, *ProfRS* algorithm reschedules the accepted requests to accommodate an urgent and high budget request. The advantage of this algorithm is that a SaaS provider accepts more users utilizing initiated VMs to earn more profit.Algorithm 2 describes *ProfRS* algorithm. In the **admission control** phase, the algorithm analyses if the new request can be accepted by waiting in an already initiated VM, inserting into an initiated VM, or initiating a new VM. Hence, firstly it verify if new request can wait all accepted requests in any already initiated VM – invoking *Wait Strategy* (Step 3). If the request cannot wait, then it checks if the new request can be inserted before any accepted request in an already initiated VM – using *Insert Strategy* (Step 4). Otherwise the algorithm checks if it can be accepted by initiating a new VM provided by any IaaS provider – using *Initiate New VM Strategy* (Step 5). If a SaaS provider does not make sufficient profit by any strategy, the algorithm *rejects* this user request (Steps 10, 11). Otherwise the algorithm gets the maximum return from all analysis results (Step 15). The remaining steps are the same as those in *ProfminVM* algorithm. The time complexity of this algorithms is $O(R\,J + R^2)$, where $R$ indicates total number of requests, $J$ indicates total number of IaaS providers.

**Algorithm 2.** Pseudo-code for *ProfRS* algorithm.

**Input:** New user's request parameters ($u^{new}$), $expInvRet^{new}_{ij}$

**Output:** Boolean

**Functions:**
***admissionControl*** {
**1.** **If** (there is any initiated VM) {
**2.** **For each** $vm_i$ in each resource provider $rp_j$ {
**3.** **If** (! ***canWait***$(u^{new}, vm_i)$) {
**4.** **If** (! ***canInsert***$(u^{new}, vm_i)$) {
**5.** **If** (! ***canInitiateNew***$(u^{new}, rp_j)$) {
**6.** **continue;**
**7.** }
**8.** }
**9.** }
**10.** **Else If** (! ***canInitiateNew***$(u^{new}, rp_j)$)
**11.** **Return *reject***
**12.** **If** (PotentialScheduleList is empty)
**13.** **Return *reject***
**14.** **Else** {
**15.** Get the ***max***$[ret^{new}_{ij}, SD_{ij}]$ in PotentialScheduleList
**16.** **If** ***(max***$(ret^{new}_{ij}) \geq expInvRet^{new}_{ij}$***)***
**17.** **Return *accept***
**18.** **Else**
**19.** **Return *reject***
**20.** }
}
}
}
***schedule*()** {
**21.** Get the $[ret^{new}_{max}, SD_{max}]$ in *maxRet*(PotentialScheduleList)
**22.** **If** ($SD_{max}$ is initiateNewVM)
**23.** initiateNewVM in $rp_j$
**24.** Schedule the $u^{new}$ in $VM_{max}$ in $rp_{max}$ according to $SD_{max}$.
}

### 3.2.2. Maximizing the profit by exploiting penalty delay (ProfPD)

To further optimize the profit, we design the algorithm *ProfPD* by considering delaying the new requests to accept more requests.Algorithm 3 describes *ProfPD* algorithm. In the **admission control** phase, we analyse if the new user request can be processed by queuing it up at the end of an already initiated VM, by inserting it into an initiated VM, or by initiating a new VM. Hence, firstly the algorithm check if the new request can wait all accepted requests to complete in any initiated VM – invoking *Wait Strategy* (Step 3). If the request cannot wait, then it checks if the new request can be inserted before any accepted request in any already initiated VM – using *Insert Strategy* (Step 4). Otherwise the algorithm checks if the new request can be accepted by initiating a new VM provided by any resource provider – using *Initiate New VM Strategy* (Step 5) or by

delaying the new request with penalty compensation – using *Penalty Delay Strategy* (Step 7). If a SaaS provider does not make sufficient profit by any strategy, the algorithm *rejects* the new request (Step 14). Otherwise, the request is accepted and scheduled based on the entry in *PotentialScheduleList* which gives the maximum return (Step 23). The rest of the steps are the same as those in *ProfminVM*. The time complexity of this algorithms is $O(R\,J + R^2)$, where $R$ indicates total number of requests, $J$ indicates total number of IaaS providers.

**Algorithm 3.** Pseudo-code for *ProfPD* algorithm.

**Input:** New user's request parameters ($u^{new}$), $expInvRet^{new}_{ij}$

**Output:** Boolean

**Functions:**
***admissionControl()*** {

**1.** **If** (there is any initiated VM) {
**2.** **For each** $vm_i$ in each resource provider $rp_j$ {
**3.** **If** (! ***canWait***($u^{new}$, $vm_i$ )) {
**4.** **If** (! ***canInsert***($u^{new}$, $vm_i$ )) {

**5.** **If** (! ***canInitiateNew***($u^{new}$, $rp_j$ ))
**6.** **continue;**
**7.** **If** (! ***canPenaltyDelay***($u^{new}$, $rp_j$ ))

**8.** **continue;**

**9.** }

**10.** }

**11.** }

**12.** }

**13.** **Else If** (! ***canInitiateNew***($u^{new}$, $rp_j$ ))
**14.** **Return *reject***

**15.** **If** (PotentialScheduleList is empty)

**16.** **Return *reject***

**17.** **Else** { Get the ***max***[$ret^{new}_{ij}$, $SD_{ij}$ ] in PotentialScheduleList
**18.** **If** (***max***($ret^{new}_{ij}$) $\geq$ $expInvRet^{new}_{ij}$)
**19.** **Return *accept***

**20.** **Else**

**21.** **Return *reject***

**22.** }

}
***schedule()*** {

**23.** Get the [$ret^{new}_{max}$, $SD_{max}$] in *maxRet(*PotentialScheduleList*)*

**24.** **If** ($SD_{max}$ is initiateNewVM)

**25.** initiateNewVM in $rp_j$
**26.** Schedule the $u^{new}$ in $VM_{max}$
}

## IV. PERFORMANCE EVALUATION

In this section, we first explain the reference algorithms and then describe our experiment methodology, followed by performance evaluation results, which includes comparison with reference algorithms and among our proposed algorithms.As existing algorithms in the literature are designed to support scenarios different to those considered in our work, we are comparing proposed algorithms to reference algorithms exhibiting lower and up bounds: *MinResTime* and *StaticGreedy*.

- The *MinResTime* algorithm selects the IaaS provider where new request can be processed with the earliest response time to avoid deadline violation and profit loss, therefore it minimizes the response time for users. Thus, it is used to know how fast user requests can be served[8].
- The *StaticGreedy algorithm* assumes that all user requests are known at the beginning of the scheduling process. In this

algorithm, we select the most profitable schedule obtained by sorting all the requests either based on *Budget or Deadline*, and then using *ProfPD* algorithm. Thus, the profit obtained from *StaticGreedy* algorithm acts as an upper bound of the maximum profit that can be generated[13]. It is clear that assumption taken in *StaticGreedy* algorithm is not possible in reality as all the future requests are not known.

### 4.1. Experimental methodology

We use CloudSim as a Cloud environment simulator and implement our algorithms within this environment. We observe the performance of the proposed algorithms from both users' and SaaS providers' perspectives.

Table 1. resource provider characteristics.

| Provider | VM types | VM price ($/hour) |
|---|---|---|
| Amazon EC2 | Small/Large | 0.12/0.48 |
| GoGrid | 1 Xeon/4 Xeon | 0.19/0.76 |
| RackSpace | Windows | 0.32 |
| Microsoft Azure | Compute | 0.12 |
| IBM | VMs 32-bit (Gold) | 0.46 |

#### 4.1.1 User's side

- In common economic models, **budget** is generated by random numbers [1]. Therefore, we follow the same random model for budget, and vary it from "very small" (mean = 0.1$) to "very large" *(*mean = 1$*)*. We choose budget factor up to 1, because the trend of results does not show any change after 1.
- Five different types of **request arrival rate** are used by varying the mean from 1000 to 5000 users per second.

We consider five resource providers – IaaS providers, which are Amazon EC2 , GoGrid , Microsoft Azure , RackSpace and IBM . To simulate the effect of using different VM types, MIPS ratings are used. Thus, a MIPS value of an equivalent processor is assigned to the request processing capability of each VM type. The price schema of VMs follows the price schema of GoGrid , Amazon EC2 , RackSpace , Microsoft Azure , and IBM . The detail resource characteristics which are used for modelling IaaS providers are shown in Table 1. The three different types of average VM **initiation time** are used in the experiment, and the mean initiation time varies from 30 seconds to 15 minutes *(*standard deviation = *(1/2)* × mean*)*. The mean of initiation time is calculated by conducting real experiments of 60 samples on GoGrid and Amazon EC2 done for four days (2 week days and 2 weekend days).

### 4.2. Performance results

In this section, we first compare our proposed algorithms with reference algorithms by varying number of users. Then, the impact of QoS parameters on the performance metrics is evaluated. Finally, robustness analysis of our algorithm is presented. All of the results present the average obtained by 5 experiment runs. In each experiment we vary one parameter, and others are given constant mean value.

The constant mean, which are used during experiment, are as follows: arrival rate = 5000 requests/sec, deadline = 2 ∗ estprocT, budget = 1$, request length is $4 \times 10^6$ MI, and penalty rate factor *(r)* = 10.

## V. CONCLUSIONS AND FUTURE DIRECTIONS

We presented scheduling algorithms for effcient resource allocation to maximize profit and cus-tomer level satisfaction for SaaS providers. Through simulation, we showed that the algorithms work well in a number of scenarios. Simulation results show that in average the *ProfPD* algorithm gives the maximum profit (in average save about 40% VM cost) among all proposed algorithms by varying all types of QoS parameters[14]. If a user request needs fast response time, *ProfRS* and *ProfminVM* could be chosen depending on the scenario.
In this work, we have assumed that the estimated service time is accurate since existing performance estimation tech-niques (e.g. analytical modelling , empirical, and historical data can be used to predict service times on various types of VMs. However, still some error can exist in this estimated service time [4] due to variable VMs' performance in Cloud. The impact of error could be minimized by two strategies: first, considering the penalty compensation clause in SLAs with IaaS provider and enforce SLA violation; second, adding some slack time during scheduling for preventing risk.

In the future we will increase the robustness of our algorithms by handling such errors dynamically. In addition, due to this performance degradation error, we will consider SLA negotiation in Cloud computing environments to improve the robustness[13]. We will also add different type of services and other pricing strategies such as spot pricing to increase the profit of service provider. Moreover, to investigate the knowledge-based admission control and scheduling for maximizing a SaaS provider's profit is one of our future directions for improving our algorithms' time complexity.

## VI. ACKNOWLEDGEMENTS

Research on market driven r esource allocation and admission control has started as early as 1981. Most if the market-based resource allocation methods are either non-pricing-based or designed for fixed number of resources. such as FirstPrice [4] and FirstProfit. In cloud , IaaS providers focusing on maximize profit and many works proposed marked based scheduling approaches.

## REFERENCES

[1] D.N. Jaideep, M.V. Varma, Learning Based Opportunistic Admission Control Algorithms For Map Reduce As A Service In: Proceedings Of The 3rd India Software Engineering Conference (ISEC 2010), Mysore, India, 2010.
[2] O.F. Rana, M. Warnier, T.B. Quillinan, F. Brazier, D. Cojocarasu, Managing Violations In Service Level Agreements, In: Proceedings Of The 5th Internationalworkshop On Grid Economics And Business Models (Gencon 2008), Gran Canaria, Spain, 2008.
[3] Y. Yemini, Selfish Optimization In Computer Networks Processing, In: Proceedings Of The 20th IEEE Conference On Decision And Control Including The Symposium On Adaptive Processes, San Diego, USA, 1981.
[4] R. Buyya, C.S. Yeo, S. Venugopal, J. Broberg, I. Brandic, Cloud Computing Emerging IT Platforms: Vision, Hype, And Reality For Delivering Computing As 5th Utility, Future Gener. Comput. Syst. 25 (6) (2009) 599–616, Elsevier Science, Amsterdam, The Netherlands.
[5] D. Parkhill, The Challenge Of The Computer Utility, Addison–Wesley, USA, 1966.
[6] M. Bichler, T. Setzer, Admission Control For Media On Demand Services. Service Oriented Computing And Application, In: Proceedings Of IEEE Internationalconference On Service Oriented Computing And Applications (SOCA 2007), Newport Beach, California, USA, 2007]
[7] K. Coleman, J. Norris, G. Candea, A. Fox, Oncall: Defeating Spikes With A Free-Market Application Cluster, In: Proceedings Of The 1st International Conference On Autonomic Computing, New York, USA, 2004.
[8] R. Buyya, R. Ranjan, R.N. Calheiros, Intercloud: Utility-Oriented Federation Of Cloud Computing Environments For Scaling Of Application Services, In: Proceedings Of The 10th International Conference On Algorithms And Architectures For Parallel Processing (ICA3PP 2010), Busan, South Korea, 2010. [9] D.A. Menasce, V.A.F. Almeida, R. Fonseca, M.A. Mendes, A Methodology For Workload Characterization Of E-Commerce Sites, In: Proceedings Of The 1999 ACM Conference On Electronic Commerce (EC 1999), Denver, CO, USA, 1999. [10] K. Xiong, H. Perros, SLA-Based Resource Allocation In Cluster Computing Systems, In:
[9] Proceedings Of 17th IEEE International Symposium On Parallel And Distributed Processing (IPDPS 2008), Alaska, USA, 2008.
[10] S.K. Garg, R. Buyya, H.J. Siegel, Time And Cost Trade-Off Management For Scheduling Parallel Applications On Utility Grids, Future Gener. Comput.Syst. 26 (8) (2009) 1344–1355.
[11] M. Islam, P. Sadayappan, D.K. Panda, Towards Provision Of Quality Of Service Guarantees In Job Scheduling, In: Proceedings Of The 6th IEEE International Conference On Cluster Computing (Cluster 2004), San Diego, USA, 2004.
[12] S. Kumar, K. Dutta, V. Mookeriee, Maximizing Business Value By Optimal Assignment Of Jobs To Resources In Grid Computing, European J. Oper.Res. 194 (3) (2009).
[13] S. Ostermann, A. Iosup, N. Yigitbasi, R. Prodan, T. Fahringer, D. Epema, A Performance Analysis Of EC2 Cloud Computing Services For Scientific Computing, In:
[14] Proceedings Of 1st International Conference On Cloud Computing (Cloudcomp), Munich, Germany, 2009.