SLA-based Admission Control for a Software-as-a-Service Provider 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, sgrag, raj}@csse.unimelb.edu.au

Abstract

With the increasing popularity of Cloud computing, the requirement for services supporting brokering across multiple infrastructure providers is growing rapidly. Cloud Computing environments are not only dynamic, but also heterogeneous with multiple types of Virtual Machine (VM) offered by various infrastructure providers. Similarly, the demand on services can also vary with time, which affects the number of VMs to be initiated. In this environment, the aim of Software as a Service (SaaS) providers is to maximize their profit and enhance their reputation by meeting Service Level Agreement (SLA) requirements of all accepted requests. SLAs are signed between SaaS providers and the customers to decide on the issues such as payment and Quality of Service (QoS). Thus, SaaS providers need effective strategies for accepting particular request, how many and what type of VMs to be initiated from suitable IaaS provider. This paper proposes admission control and scheduling algorithms that take into account dynamic parameters such as variation in VM's initiation time and user's QoS requirements such as budget deadline, and penalty rate ratio. This paper also presents an extensive evaluation study to analyse well suited algorithm for a particular scenario to maximise the SaaS provider's profit.

Keywords: *Cloud computing; Service Level Agreement (SLA); Admission Control; Software as a Service; Scalability of Application Services*

I. INTRODUCTION

Cloud computing has emerged as a new paradigm for offering elastic access to dynamically provisioned IT resources on a pay-as-you-go model to multiple customers. It has been increasingly adopted in many areas, such as banking, e-commerce business, retail industry and academy due to its flexibility, scalability and cost-effectiveness [8][9][11]. Cloud computing includes Software (SaaS), Platform (PaaS) and Infrastructure (IaaS) as a Service. This paper focuses on SaaS providers, who lease VMs provided by IaaS providers, to offer software services. For example, Animoto provides automated video generation with images using Amazon Cloud infrastructure services [23].

The main objective of SaaS providers is to maximize their profit by either attracting large market share or minimizing the cost. Cloud computing provides a very effective platform for enhancing market share. The Cloud providers generally list their customers' applications on their web sites, which acts as an indirect advertisement for their customers. For instance, Amazon's Customer App Catalogue includes all the details about the customers' applications. Another way to enlarge market share is to improve reputation by accepting more user requests and improve user satisfaction level by meeting the contract terms defined in an SLA [40]. To minimize the cost, a SaaS provider has to deal with the following questions:

- How many and which type of VMs are required from an IaaS provider?
- How to map user requests to VMs?

Therefore, to maximize profit, the SaaS provider needs to satisfy a major number of users while minimizing its cost. To satisfy a large number of users, the SaaS provider requires initiating more VMs. However, currently IaaS providers allow only a limited number of VMs to be initiated by a user [33]. For instance, Amazon [33] and GoGrid [35] allow only the instantiation of maximum 200 VMs. Thus, using only one IaaS provider, SaaS providers limit their capability to simultaneously serve several users. Hence, it is essential for a SaaS provider to leverage multiple IaaS providers. In this case, a SaaS provider has to tackle many more challenges to minimize cost, such as:

- How many IaaS providers are required?
- Which IaaS provider should be chosen?
- How many VMs are required from each IaaS provider?

In addition, while minimizing the cost, a SaaS provider has to manage the multiple users' demand and contractual obligations in the form of SLAs. In other words, it has to balance between the cost minimization and satisfaction of the SLA requirements. Moreover, multiple IaaS providers can offer many different types of VMs with various

pricing schemes and cloud interfaces. In any case, with the profit maximization mechanisms, a SaaS provider requires to leverage multiple user requests and multiple IaaS providers. In order to utilizing resources effectively and efficiently, admission control and scheduling capabilities at the platform layer to deal with various user requirements and heterogeneity at the infrastructure level. A high level system model for application service scalability using multiple IaaS providers in Cloud is illustrated in Figure 1. A user sends requests for utilizing software applications offered by a SaaS provider, who includes two infrastructure layers, namely application layer and platform layer, to satisfy the user's request. The application layer controls all application services which a SaaS provider can offer to users. The platform layer includes admission control and scheduling policies for making decision whether to reject or accept a request, and respond to the user's request. In addition, the platform layer also schedules the processing of requests on VMs from IaaS providers. In academy and industry, pure PaaS maps to our scenario's platform layer.

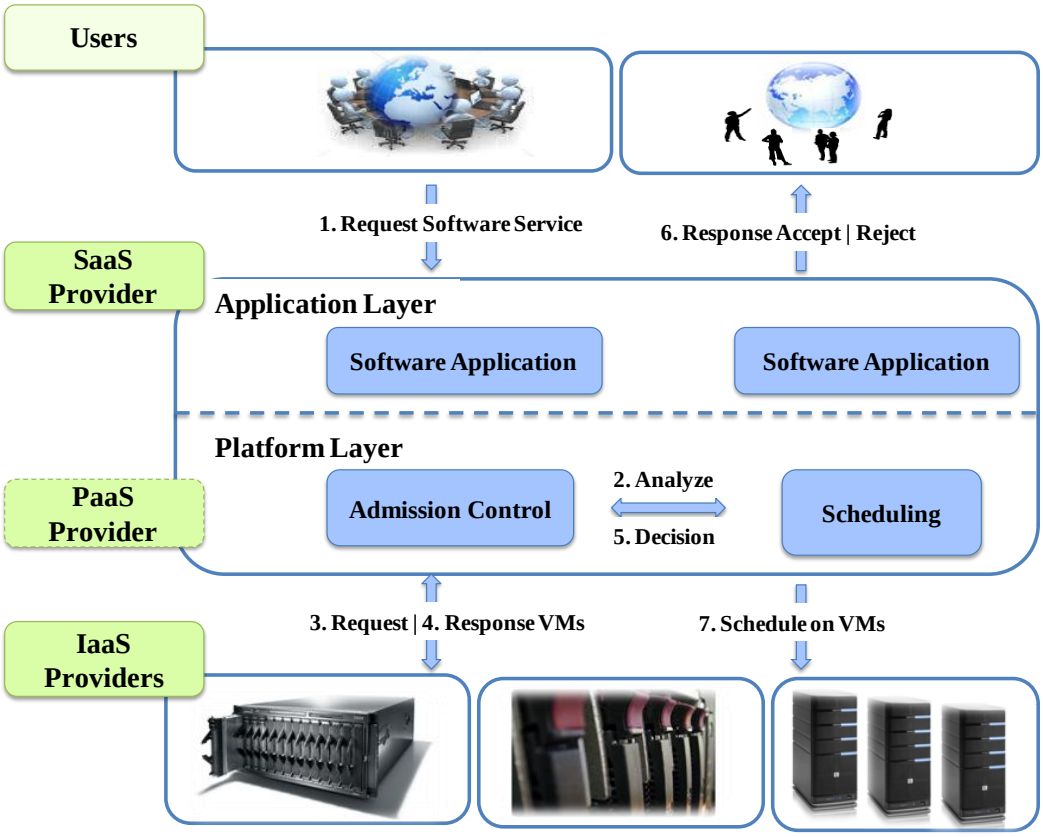


Figure 1. A high level system model for application service scalability using multiple IaaS providers in Cloud.

In this context, in order to achieve SaaS provider's objective of profit maximization, our work proposes scheduling mechanisms which minimize the SaaS providers' cost by optimizing the placement of virtual infrastructures across multiple IaaS providers and also provide the guaranteed QoS to users. Projects such as InterCloud [17], Sky Computing [19], and Reservoir [18] investigated the technological advancement that is required to aid the

deployment of cloud services across multiple infrastructure providers. Thus, times are mature to examine the admission control and scheduling strategies, which allow for a cost effective usage of physical resources in Clouds. Although many works [30][16][3] proposed market based scheduling approaches to maximize the profit of IaaS providers, research at the SaaS provider level in the context of Cloud computing is still in its infancy. Many works do not consider the leasing scenario from multiple IaaS providers, where physical resources can be dynamically expanded and contracted based on demand.

Therefore, in this paper, we propose cost effective admission control and scheduling mechanisms with the goal of maximizing the SaaS provider's profit, by optimizing the placement of customer's service request on the VMs leased from multiple IaaS providers. Our mechanisms take into account customer's QoS requirements such as deadline, SLA requirements, and infrastructure heterogeneity in terms of VM types, VM initiation time, data transfer time and pricing policies. The key contributions of this paper are the following:

- It defines two layers of SLAs - SLA(U) with users and SLA(R) with IaaS providers based on eight basic QoS parameters.
- It designs and implements admission control and scheduling mechanisms to maximize service provider's profit. The admission control mechanism examines which user request be accepted with least effect on other accepted requests; especially, how incurred penalties will decrease profit. The scheduling mechanism determines where and which type of VM will be initiated by incorporating the heterogeneity of IaaS providers in terms of their price, dynamic VM initiation time, dynamic service time, and data transfer time.
- It presents performance analysis of the proposed algorithms based on varying SLA (U) properties: (i) deadline, (ii) arrival rate, (iii) budget, (iv) service time, (v) penalty rate ratio, and (vi) input file size; and SLA(R) property: (i) initiation time. Moreover, we also examine algorithms' robustness by considering violation of SLA(R).

The rest of this paper is organized as follows. In Section II, we discuss prior works in SLA, which have been carried out in Cluster, Grid, and Cloud computing contexts. We also identify how the work is differs from related works. Section III presents the detailed scenario and outlines the two layers of SLAs supporting QoS parameters. Section IV presents four analysis strategies used in the proposed algorithms, and describes the proposed algorithms, which are *ProfminVM*, *ProfRS* and *ProfPD*. Section V firstly presents the experimental methodology including test bed and evaluation metrics; secondly, discusses the overall comparison of the performance evaluation results with reference algorithms; thirdly, compares the algorithms by providing the insights on when to use each algorithm when each algorithm should be used; fourthly, evaluates the robustness of the algorithms. Finally, Section VI concludes the paper by summarizing the comparison results and future work.

II. RELATED WORK

Research on market driven resource allocation and admission control has started as early as 1981 [10][6]. Most of the market-based resource allocation methods are either non-pricing-based [16] or designed for fixed number of resources, such as FirstPrice [4] and FirstProfit [7]. Our work is related to profit driven SLA based on admission control and scheduling in Cloud computing environments. Thus, in following sections, we describe the differences between our and the most relevant previous works.

Cluster & Grid Computing

Chun et al.[14] built a prototype cluster of time-sharing CPU usage to serve user requests. Another market-based approach to solve traffic spikes for hosting Internet applications on Cluster was studied by Coleman et al. [14]. Their common concern is fixed number of resources. However, our work focuses on multiple resource providers which supply dynamic number of VMs, where the resources are distributed globally and can be expanded on demand in Cloud.

Liu et al. [24] analysed the problem of maximizing profit in e-commerce environment using web service technologies, where the basic distributed system is Cluster. Their methodology belongs to general resource management with SLAs that includes the "tail distributions of the per-class delays, per-class throughputs and mean delays" [24]. However, we consider budget, deadline as QoS parameters, which are different from their work. Besides resource management, admission control is also our main focus.

Menasce et al. [25] proposed a priority schema for requests during scheduling based on the user status, such as normal navigation or shopping. Authors believe that users will lose patience if response time is too long. Hence, their policy prioritised the request and assigned higher priority to requests with shopping status during scheduling to improve the revenue. Nevertheless, response time is only the aspect of the concern. In addition, their work is not SLA-based.

Xiong et al. [28] focused on SLA-based resource allocation in Cluster computing systems, where QoS metrics considered are response time, Cluster utilization, packet loss rate and Cluster availability. The main differences between their work and ours are that we consider different QoS parameters (i.e., budget, deadline, penalty rate, admission control and resource allocation, and multiple IaaS providers.

Yeo and Buyya [1] presented algorithms to handle penalties in order to enhance the utility of the cluster based on SLA. They have outlined a basic SLA with four parameters in cluster environment. Our current proposed work is quite different from that previous work because here we investigate resource outsourcing model from multiple IaaS providers in Clouds. Moreover, in the previous work, we only consider one SLA layer (between user and resource provider) while here we focus on two layers of SLAs, which are established with both end users and IaaS providers.

Kumar et al. [41] investigated two heuristics, HRED and HRED-T, to minimize business value, such as cost or time, of users. They studied only the minimization of cost. Garg et al. [30] also proposed time and cost based resource allocation in Grids on multiple resources for parallel applications. However, our current studies use different QoS parameters, (e.g. penalty rate). In addition, our current studies focus on Clouds, where the unit of resource is mostly VM, which may consist of multiple processors.

Bichler and Setzer [13] proposed the admission control for media on demand services, where the duration of service is fixed in their scenario. The system accepts a request if its revenue is more than its cost. Our approachs allow a SaaS provider to specify its expected profit ratio according to the cost, for example; the SaaS provider can specify that the service request which can gain 3 times profit over total cost can be accepted.

Netto et al. [29] considered deadline as a QoS parameter for bag-of-task applications in utility computing systems. They considered multiple providers but only focused on deadline constraint, which is one of QoS parameters we are focusing on in the current proposed work.

Islam et al. [31][32] have investigated policies for admission control that consider jobs with deadline constraints and response time guarantees. The main difference is that they consider parallel jobs submitted to a single site, whereas we utilize multiple VM from multiple IaaS providers to serve multiple requests.

Cloud Computing

Reig G. et al [27] contributed on minimizing the resource consumption by 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 request before its deadline. Our works similarity is that both the works use the deadline constraint to reject some requests for more efficient scheduling. However, in our work deadline is only one of the constraints, which is used to optimize resource scheduling; and we also consider the profit constraint to avoid wastage of resources on low profit requests.

Popovici et al. [7] mainly focused on QoS parameters on resource provider's side such as price and offered load. However, proposed work differs on QoS parameters from both users' and SaaS providers' point of view, such as budget, deadline and penalty rate.

Jaideep and Varma [2] proposed learning-based admission control in Cloud computing environment. For instance, learning-based opportunistic algorithm admits requests only if they are unlikely to cross the overload threshold set by the service provider. Thus, this work focuses on the accuracy of admission control but does not consider software service providers' profit.

Lee et al. [3] investigated the profit driven service request scheduling for workflow. The context of this work differs from our proposed work in many ways, for instance, our work a) focus on SLA driven QoS parameters on both user and provider sides, such as initiation time, and b) considers heterogeneity between multiple resource providers in terms of VM type and price offerings.

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

- **The utility function is time-varying by considering dynamic VM deploying time (*aka* initiation time), processing time and data transfer time.**
- **It adapts to dynamic resource pools and consistently evaluates the profit of adding a new instance or removing instances, while most previous work deal with fixed size of resource pools.**

A summary of comparison between related and our work in Cloud Computing is given below in Table 1:

III. SYSTEM MODEL

As discussed previously, the system model considered is based on the Cloud computing environment, whereby Cloud users want to perform some tasks using application service provided by a SaaS provider, who leases infrastructure from multiple IaaS providers to deploy the software services. The SaaS provider's objective is to accept and schedule a user request such that its profit is maximised while the Quality of Service (QoS) requirements of user are assured. Users request the software from a SaaS provider by submitting their QoS requests and input files. The platform layer of SaaS provider uses **admission control** mechanisms to interpret and analyse the user's QoS parameters and decides whether to accept or reject the request based on the capability and availability of VMs. Then, **scheduling** mechanisms facilitate the SaaS provider's platform layer to allocate resources based on the decision of admission control.

Table 1. The summary and comparison of the related works in Cloud Computing area.

Related Works	*Admission Control*	*Resource Management (Scheduling)*	*Profit Driven*	*Resource Characteristics*	*SLA Oriented*	*QoS*	
						For Users	**For SaaS Providers**
Reig G. et al [27]	Yes	Yes	No	NA	Yes	Deadline	No
Bichler and Setzer [13]	Yes	No	Yes	NA	Yes	Budget	No
Jaideep and Varma [2]	Yes	No	No	One resource provider	No		No
Lee et al. [3]	Yes	Yes	Yes	One resource provider	Yes		NA
Wu et al. (proposed work)	Yes	Yes	Yes	Multiple resource providers	Yes	Deadline. Budget. Request length. Penalty Rate.	VM Initiation Time. Data Transfer Time.

Actors

The participating parties involved in the process are discussed below along with their objectives and constraints:

A. 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 of SaaS provider utilizes the admission control and scheduling mechanisms 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. SLA with Users – SLA (U) includes the following properties:

- **Deadline:** Maximum time user would like to wait for the result.
- **Budget:** How much user is willing to pay for the requested services.
- **Penalty Rate Ratio:** A ratio for consumers' compensation if SaaS providers miss the deadline.
- **Input File Size:** The size of input file provided by users.
- **Request Length:** How many Millions of Instructions (MI) are required to be executed to serve the request.

B. Resource Provider

A Resource provider (*RP*) or 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. In this paper, we consider multiple resource providers, because from a SaaS provider's point of view, one resource provider may not be able to offer all required resources to assure SLA (U). In addition, the service provider can exploit the advantages of one resource provider over others to satisfy the constrained user requests. For example, one resource provider may offer VMs with high processing capability, but charging a higher price. The service provider can use such resources to satisfy urgent user requests.

It is important to establish SLA with a resource provider – SLA (R), because it enforces the resource provider to guarantee service quality. Furthermore, it provides a risk transfer for SaaS providers, when the terms are violated by resource provider. In this work, we do not consider the compensation given by the resource provider because 85% resource providers do not really provide penalty enforcement for SLA violation currently [34]. The SLA (R) includes the following properties:

- **Initiation Time**: How long it takes to deploy a VM.

- **Price**: How much a SaaS provider has to pay per hour for using a VM from a resource provider?
- **Input Data Transfer Price:** How much a SaaS provider has to pay for data transfer from local machine to resource provider's VM.
- **Output Data Transfer Price**: How much a SaaS provider has to pay for data transfer from resource provider's VM to local machine?
- **Processing Speed**: How fast the VM can process?. We use Machine Instruction Per Second (MIPS) of a VM as processing speed.
- **Data Transfer Speed**: How fast the data is transferred.? It depends on the location distance and also the network performance.

C. SaaS provider

A SaaS provider leases resources from IaaS providers and in turn leases software as services to end users. SaaS providers aim at minimizing the cost of using resources from IaaS providers, and therefore maximizing the net profit earned through serving user requests. It is also in the interest of SaaS providers to guarantee QoS levels of accepted users in order to enhance their reputation. From SaaS provider's point of view, there are two layers of SLA, which are SLA (U) and SLA(R). If any party in the contract violate its terms, the defaulter has to pay for the penalty according to the clauses defined in the SLA.

Mathematical Models

A. Penalty model

We model the SLA violation penalty (P) as linear function which is similar to other related works [1][4][5].

$$P = \alpha + \beta \times DT \qquad \text{where } \beta \text{ is the penalty rate and } DT \text{ is delay time.} \tag{1}$$

The penalty function not only penalizes the service provider by reducing the utility (profit) but also compensates the users for tolerating the service failure. Thus, the penalty function is designed such that it reduces the budget of the job over time after lapse of its deadline. Figure 2 shows the impact of penalty rate β on the SaaS provider's utility. In the figure, "Delay" is the tolerance time after which budget will become zero.

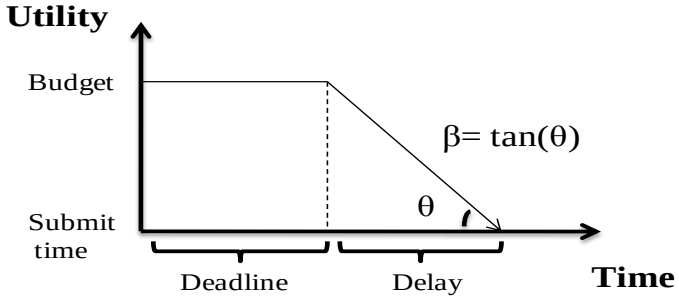


Figure 2. Impact of penalty function on utility

B. Profit Model

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* provides 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 $inPri_j$ *and* $outPri_j$ *respectively*. Let ($iniT_{ij}$) 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* β^{new}. Let $inDS^{new}$ and $outDS^{new}$ be the data-in and –out required to process the user requests.

Let $Cost_{ijl}^{new}$ 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 $Prof_{ij}^{new}$ gained by the SaaS provider is defined as:

$$Prof_{ij}^{new} = B^{new} - Cost_{ij}^{new} \; ; \forall i \in I, j \in J \tag{2}$$

The total cost incurred to SaaS provider for accepting the new request consists of request's processing cost (PC_{ij}^{new}), data transfer cost (DTC_j^{new}), VM initiation cost (IC_{ij}^{new}), and penalty delay cost (PDC_{ij}^{new}) (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*:

$$Cost_{ij}^{new} = PC_{ij}^{new} + DTC_j^{new} + IC_{ij}^{new} + PDC_{ij}^{new} \; ; \forall i \in I, j \in J, l \in N_j \tag{3}$$

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

$$PC_{ij}^{new} = procT_{ij}^{new} \times P_{jl}, \forall i \in I, j \in J, l \in N_j \tag{4}$$

Data transfer cost as described in Equation (5) includes cost for both data-in and data-out.

$$DTC_j^{new} = inDS^{new} \times in\mathrm{Pri}_j + outDS^{new} \times out\mathrm{Pri}_j \; ; \forall j \in J \tag{5}$$

The initiation cost (IC_{ij}^{new}) of VM *i* (type *l*) is dependent on the type of VM initiated in the datacenter of IaaS provider *j*.

$$IC_{ij}^{new} = iniT_{ij} \times P_{jl}, \forall i \in I, j \in J, l \in N_j \tag{6}$$

In Equation (7), penalty delay cost (PDC_{ij}^{new}) is how much the service provider has to give discount to users for SLA(U) violation. It is dependent on the penalty rate (β^{new}) and penalty delay time (PDT_{ij}^{new}) period.

$$PDC_{ij}^{new} = \beta^{new} \times PDT_{ij}^{new} \; ; \; \forall i \in I, j \in J \tag{7}$$

To process any new request, SaaS provider either initiate a new VM or schedule the request on already initiated VM. If service provider schedules the new request on 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 till it start processing on VM i is $\sum_{k=1}^{K} procT_{ij}^{k}$,where K is the number of request yet to be processed before the new request. Thus, PDT_{ij}^{new} is given by:

$$PDT_{ij}^{new} = \begin{cases} t + \sum_{k=1}^{K} procT_{ij}^{k} + procT_{ij}^{new} - DL^{new}, & if\ new\ VM\ is\ not\ initiated \\ procT_{ij}^{new} + iniT_{ij} + DTT_{ij}^{new} - DL^{new}{}_{ij}, & if\ new\ VM\ is\ initiated \end{cases} \quad (8)$$

Where, DTT_{ij}^{new} is the data transfer time which is the summation of time taken to upload the input ($inDT_{ij}^{new}$) and download the output data ($outDT_{ij}^{new}$) from the VM i on IaaS Provider j. The data transfer time is given by:

$$DTT_{ij}^{new} = inDT_{ij}^{new} + outDT_{ij}^{new}; \forall i \in I, j \in J \quad (9)$$

Thus, the response time (T_{ij}^{new}) for the new request to process it on VM i of IaaS Provider j is calculated in Equation (10) and consists of VM initiation time ($iniT_{ij}^{new}$), request's service processing time ($procT_{ij}^{new}$), data transfer time (DTT_{ij}^{new}), and penalty delay time (PDT_{ij}^{new}).

$$T_{ij}^{new} = \begin{cases} \sum_{k=1}^{K} procT_{ij}^{k} + procT_{ij}^{new}, & if\ new\ VM\ is\ not\ initiated \\ procT_{ij}^{new} + iniT_{ij} + DTT_{ij}^{new}, & if\ new\ VM\ is\ initiated \end{cases} \quad (10)$$

The investment return (ret_{ij}^{new}) to accept new user request per hour on a particular VM i in IaaS Provider j is calculated based on the profit ($prof_{ij}^{new}$) and time (T_{ij}^{new}):

$$ret_{ij}^{new} = \frac{prof_{ij}^{new}}{T_{ij}^{new}} \quad ; \forall i \in I, j \in J \quad (11)$$

IV. ALGORITHM

As discussed, our main objective is to maximize the profit of a SaaS provider by minimizing the cost and maximizing the number of accepted users. To achieve it, we propose three admission control and scheduling algorithms. For admission control, the SaaS provider's platform layer first uses "**analysis strategies"** to analyze the possible scheduling decision. These QoS based analysis strategies are **a) initiate a new VM, b) queue up the new user request at the end of scheduling queue of a VM, c) insert (prioritize) new user request at the proper position before the accepted user requests and, d) delay new user request to wait all accepted users to finish**. The corresponding analysis strategies are described as follows.

Analysis Strategies

A. Initiate New VM Strategy

Function 1 describes the pseudo-code for "*initiate new VM strategy"*, where inputs are new user request QoS parameters (such as deadline, budget, penalty rate ratio, request length, input file size) and parameters related to resource provider (rp_j). Function *canInitiateNewVM* () first checks for each type of VMs in the datacenter of rp_j whether the deadline of new request is long enough for itself to complete (Step 1). The estimated finish time depends on the estimated start time, request processing time and VM initiation time.

If new request can be completed before deadline, then we calculate the investment return (Equation 11, Step2) and record all related information (such as resource provider number, VM number, start time and estimated finish time) into schedule decision (Step 3). In addition, they are recorded into potential schedule list (Step 4) and this function returns true (Step 5) which means new VM can be created. Otherwise, this function returns false (Step 6-8) which means new VM cannot be initiated.

Function 1: Pseudo-code for *Initiate New VM Strategy*

Input: New user's request parameters (u^{new}), rp_j

Output: Boolean

Function:

caniInitiateNewVM(u^{new}, rp_j ***)*** {

1. **For each** VM type available in Datacenter of rp_j {

2 **If** (u^{new} 's deadline ≥ u^{new} 's estimated finish time) {

3. Calculate the return ret_j^{new} on new initiated *VM i*

4. Record Schedule Decision SD_{ij},

5. Insert [ret_{ij}^{new}, SD_{ij}] in PotentialScheduleList

6. continue

7. **}**

8. **Else**

9. **Return *False***

10. }

}

B. Wait Strategy

Function 2 describes the pseudo-code for *wait strategy*, in which the inputs are same as the strategy 1. Function *canWait* () first checks if the deadline of new request is long enough for the request to complete (Step 1), and then checks whether the flexible time (fT_{ij}^{new}) of new request is enough to wait all accepted requests in vm_i to complete (Step 2). The fT_{ij}^{new} is given by Equation (12), in which K indicates total number of all accepted requests.

$$fT_{ij}^{new} = DL^{new} - \sum_{k=1}^{K} procT_{ij}^{k} - subT^{new}; \forall i \in I, j \in J, k \in K \quad (12)$$

If new request can wait for all accepted requests to complete, then we calculate the investment return (Step 3) and record all related information (such as resource provider number, VM number, start time and estimated finish time) into schedule decision (Step 4). In addition, they are recorded into potential schedule list (Step 5) and this function returns true (Step 6). Otherwise, this function returns false (Step 11 and 12).

Function 2: Pseudo-code for *Wait Strategy*

Input: New user's request parameters (u^{new}), vm_i

Output: Boolean

Function:

canWait (u^{new}, vm_i ***)*** {

1. **If** (u^{new} 's deadline ≥ u^{new} 's estimated finish time) {

2. **If** (u^{new} can wait all accepted users in vm_i to finish) {

3. Calculate the return ret_{ij}^{new}

4. Record Schedule Decision SD_{ij},

5. Insert [ret_{ij}^{new}, SD_{ij}] in PotentialScheduleList

6. **Return *True***

7. }

8. **Else**

9. **Return *False***

10. }

11. **Else**

12. **Return *False***

}

C. Insert Strategy

Function 3 describes the pseudo-code for "*insert strategy"*, in which inputs are the same as Function 1. Function *canInsert* () first checks whether any accepted request u_k according to latest start time in vm_i can wait new request to finish. If the flexible time of accepted request (fT_{ij}^{k}) is enough to wait for a new user request to complete (Step 1) then insert new request before request *k*. The fT_{ij}^{k} indicates duration of the request wait time with deadline and it is given by Equation (13), in which DL^k indicates the deadline of accepted request, *k* indicates the position of accepted request, and *K* indicates total number of all accepted user requests.

$$fT_{ij}^{k} = DL^{k} - \sum_{\substack{n=1,\\ n\neq k}}^{K} procT_{ij}^{n} - T_{ij}^{new} - subT^{new}; \forall i \in I, j \in J, k \in K \tag{13}$$

If there is u^k that is able to wait for new user request to complete, the algorithm checks whether the new user request can complete processing before deadline (Step 2). If so, u^{new} gets priority over u^k, then the algorithm calculates the investment return (Step 3) and records all related information (such as, **insert position k for new user**) into schedule decision (Step 4). The other remaining steps are the same as those in Function 1.

Function 3: Pseudo-code for *Insert Strategy*

Input: New user's request parameters (u^{new}), vm_i

Output: Boolean

Function:

canInsert(u^{new}, vm_i) {

1. **If** (any accepted user u^k in vm_i can wait u^{new} to finish) {

2. **If** (u^{new} 's deadline ≥ u^{new} 's estimated finish time) {

3. Calculate the return ret_{ij}^{new}

4. Record new request's insert position *k* into Schedule Decision SD_{ij}

5. Insert [ret_{ij}^{new}, SD_{ij}] in PotentialScheduleList

6. **Return *True***

7. }

8. **Else**

9. **Return *False***

10. }

11. **Else**

12. **Return *False***

}

D. Penalty Delay Strategy

Function 4 describes the pseudo-code for *penalty delay strategy*, in which inputs are the same as Function 1. Function *canPenaltyDelay*() first checks whether new user request's budget is enough to wait for all accepted user requests in vm_i to complete with delay after its deadline (Step 1). Equation (2) is used to check whether budget is enough to compensate the penalty delay loss. If penalty delay loss is affordable to a SaaS provider to accept new user request (*Profit* > 0), then we calculate the investment return (Step 2) and record all related information (such as resource provider number, VM number, start time and estimated finish time) into schedule decision (Step 3). The other remaining steps are the same as those in Function 1.

Function 4: Pseudo-code for *Penalty Delay Strategy*

Input: New user's request parameters (u^{new}), vm_i

Output: Boolean

Function:

canPenaltyDelay(u^{new} , vm_i) {

1. **If** (u^{new} can wait all accepted users with penalty delay) {

2. Calculate the return ret_{ij}^{new}

3. Record Schedule Decision SD_{ij}

4. Insert [ret_{ij}^{new}, SD_{ij}] in PotentialScheduleList

5. **Return *True***

6. }

7. **Else**

8. **Return *False***

}

Proposed Algorithms

Based on above strategies we propose three algorithms, which are *ProfminVM*, *ProfRS*, and *ProfPD:*

1. Maximizing the profit by minimizing the number of VMs (*ProfminVM*).
2. Maximizing the profit by rescheduling (*ProfRS*).
3. Maximizing the profit by exploiting the penalty delay (*ProfPD*).

These algorithms are discussed below:

A. Maximizing the Profit by Minimizing the number of VMs (ProfminVM)

A service provider can maximize the profit by reducing the resource cost, which depends on the number and type of initiated VMs in IaaS provider's datacenter. Therefore, *ProfminVM* algorithm is designed to minimize the number of VMs by maximizing the utilization of already initiated VMs. Algorithm 1 describes the *ProfminVM* algorithm, which involves two main phases: **a) admission control** and **b) schedule.**

In **admission control phase**, we analyse whether the new user request is able to be processed either by queuing it up in an already initiated VM or by initiating a new VM. Hence, firstly, it searches on which VM the new request is able to be queued by using *Wait Strategy* (Function 2). If this request cannot wait then we check whether it can be accepted by initiating a new VM provided by any IaaS provider using *Initiate New VM Strategy* (Step 3-7). If a SaaS provider does not make sufficient profit by initiating a new VM, the algorithm will reject this request (Step 8- 10). Otherwise, the algorithm gets the maximum return from all analysis results (Step 12). The acceptance and rejection of a new request depends on the maximum expected investment return ($expInvRet_{ij}^{new}$) which the SaaS provider can get. If the investment return ret_{ij}^{new} is more than the SaaS provider's $expInvRet_{ij}^{new}$, the algorithm *accepts* the new request (Step 14, 15), otherwise, the algorithm *rejects* this request (Step 16, 17). The expected investment return ratio w is customized by SaaS providers. The expected investment return ($expInvRet_{ij}^{new}$) is given by Equation (14):

$$\text{expInvRet}_{ij}^{new} = \omega \times \frac{Cost_{ij}^{new}}{T_{ij}^{new}} ; \forall i \in I, j \in J \tag{14}$$

The **schedule phase** is the actual scheduling process based on the admission control result; if the algorithm accepts the new user request, the algorithm gets the scheduling decision from the PotentialSchedulingList for which SaaS provider is making maximum profit, in other word, we first find out in which IaaS Provider rp_j and which VM vm_i a SaaS provider can gain the maximum investment return (Step 20). If the maximum investment return is gained by initiating a new VM, then the algorithm initiates a new VM in the referred rp_j, and allocates this VM to this request. Otherwise, the algorithm schedules the new request on the referred vm_i in rp_j (Step 21-23). The time complexity of this algorithm is $O(RJ+R)$, where R indicates the total number of requests and J indicates the number of IaaS providers.

Algorithm 1. Pseudo-code for *ProfminVM* algorithm

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

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** (**! canInitiateNew**(u^{new}, rp_j)) {

5. **continue;**

6. }

7. }

8. **Else If** (**!** ***canInitiateNew***(u^{new}, rp_j))

9. **Return** ***reject***

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

11. **Return** ***reject***

12. **Else** {

13. Get the ***max***[ret_{ij}^{new}, SD_{ij}] in PotentialScheduleList

14. **If** (***max***(ret_{ij}^{new}) $\geq$ $expInvRet_{ij}^{new}$)

15. **Return** ***accept***

16. **Else**

17. **Return** ***reject***

18. }

19. **}**

}

schedule ( ) {

20. Get the [ret_{max}^{new}, 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}.

}

B. Maximizing the Profit by Rescheduling (ProfRS)

In *ProfminVM* algorithm, a new user request does not get priority over any accepted requests. This inflexibility affects the profit of a SaaS provider since many urgent and high budget service requests will be rejected. Thus, the algorithm reschedules the accepted users in *ProfRS* algorithm to accommodate an urgent and high budget request. The advantage of this algorithm is that a SaaS provider accepts more users utilizing initiated VMs and earns more profit. Algorithm 2 describes *ProfRS* algorithm.

In **admission control phase**, the algorithm analyses whether the new user request is able to be processed by queuing it up in an already initiated VM, inserting it into an initiated VM, or initiating a new VM. Hence, firstly it searches on which VM the new request is able to be queued by invoking *Wait Strategy* (Function 2, Step 3). If the new request cannot wait, then the algorithm checks whether it can be accepted by inserting into already initiated VMs using *Insert Strategy* (Step 4). Otherwise, the algorithm checks whether 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 strategies, the algorithm *rejects* this user request (Step 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 $O(RJ+R^2)$, 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}), $expInvRet_{ij}^{new}$

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_{ij}^{new}, SD_{ij}] in PotentialScheduleList

16. **If** (***max***(ret_{ij}^{new}) $\geq$ $expInvRet_{ij}^{new}$)

17. **Return** ***accept***

18. **Else**

19. **Return** ***reject***

20. }

}

schedule ( ) {

21. Get the [ret_{max}^{new}, 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}.

}

C. Maximizing the Profit by exploiting penalty delay (ProfPD)

The profit of the service provider can be further enhanced by delaying the requests. In other word, a SaaS provider tries to delay a new request with penalty compensation. Algorithm 3 describes *ProfPD* algorithm.

In **admission control** phase, we analyse whether the new user request is able to be processed by queuing it up in the end of an already initiated VM, inserting it into an initiated VM, or initiating a new VM. Hence, firstly it searches on which VM new request is able to be queued by using *Wait Strategy* (Step 3). If the new request cannot wait, then the algorithm checks whether it can be accepted by inserting into already initiated VM using *Insert Strategy* (Step 4). Otherwise, the algorithm checks whether it can be accepted by initiating a new VM provided by any IaaS provider using *Initiate New VM Strategy* or by delaying the new request with penalty compensation using *Penalty Delay Strategy* (Step 5, 6). If a SaaS provider does not make sufficient profit by any strategies, we *reject* this user request (Step 16). Otherwise, the request is accepted and scheduled based on the entry in *PotentialScheduleList* which gives the maximum return (Step 17). The rest of the steps are the same as those in *ProfminVM*. The time complexity of this algorithms is O ($RJ+R^2$), 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}), $expInvRet_{ij}^{new}$

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_ij^new, SD_ij ] in PotentialScheduleList
18.       If ( max(ret_ij^new) ≥ expInvRet_ij^new )
19.           Return accept
20.       Else
21.           Return reject
22.   }
}
```

schedule ( ) {

```
23.   Get the [ret_max^new, SD_max ] in maxRet(PotentialScheduleList)
24.   If ( SD_max is initiateNewVM)
25.       initiateNewVM in rp_j
26.   Schedule the u^new in VM_max in rp_max according to SD_max.
}
```

V. PERFORMANCE EVALUATION

In this section, we present the performance results obtained from an extensive set of experiments. Since, none of the existing algorithm can be directly applied in our scheduling scenario; we used two following reference algorithms, *MinResTime* and *StaticGreedy*, to compare with our proposed algorithms.

- The *MinResTime* algorithm selects the IaaS provider where user 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.
- The *StaticGreedy algorithm,* as name suggests, assumes that all user requests are known at the beginning of the scheduling process. In this algorithm, we select the most profitable schedule obtain by sorting all the requests either based on *Budget or Deadline*, and then scheduling using *ProfPD* algorithm. Thus, the profit obtained from *StaticGreedy* Algorithm acts as an upper bound for the maximum profit generated by our three proposed algorithms. It is clear that assumption taken in *StaticGreedy* Algorithm is not possible in reality as we cannot know all the future user requests.

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 results showing the impact of **users' side** QoS parameters **(i) request arrival rate, (ii) deadline, (iii) budget, (iv) service time, and (v) penalty rate ratio; IaaS providers' side** QoS parameter **(i) VM initiation time;** and robustness analysis of our algorithms**.**

Experimental Methodology

CloudSim [20] is used to simulate a cloud computing environment that utilises our proposed algorithms for admission control and resource allocation. We observe the performance of the proposed algorithms from both users' and SaaS providers' perspectives. From users' perspective, we observe how many requests are accepted and how fast user requests are processed (we call it average response time). From SaaS providers' perspective, we observe how much profit they gain and how many VMs they initiate. Therefore, we use four performance measurement metrics: total profit, average request response time, number of initiated VMs, and number of accepted user. All the parameters from both users' and IaaS providers' side used in the simulation study are given in following section:

A) Users' side: We examine our algorithms with 5000 users. From user side, five parameters (deadline, service time, budget, arival rate and penalty rate factor) are varied to evaluate their impact on the performance of our

proposed algorithms. Since there is no available workload specifiying these parameters, thus, we use normal distribution (standard deviation =(1/2)xmean) to model all parameters, except request arrival rate, which follows poisson distribution.

- The equation to calculate **deadline** mean (DL_{ij}^{new}) is given in Equation 15. α is the factor which is used to vary the deadline from "very tight" (α =0.5) to "very relax" (α =2.5). $estprocT_{ij}^{new}$ indicates the new service request's estimated processing time.

$$DL_{ij}^{new} = \alpha \times estprocT_{ij}^{new} + estprocT_{ij}^{new}; \forall i \in I, j \in J \quad (15)$$

- **Service time** is estimated based on the Request Length (*MI*) and the Millions of Instruction per Second (*MIPS*) of a VM. The mean Request Lengths are selected between 10^6MI ("very small") to $5x10^6$MI ("very large"), while *MIPS* value for each VM type is fixed.
- 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.
- The penalty rate β (the same as in Equation 1) is modelled by Equation 16. It is calculated in terms of how long a user is willing to wait (r) in proportion to the deadline when SLA is violated. In order to vary the penalty rate, we vary the mean of r from "very small" (4) to "very large" (44).

$$\beta = \frac{B^{new}}{DL^{new} \times r}; \forall i \in I, j \in J \quad (16)$$

B) Resource Providers' side:

We consider five 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, the request processing capability of each VM type is assigned a MIPS value of an equivalent processor. The **prices for** VM follows the price schema of GoGrid [35] , Amazon EC2 [33], RackSpace [36], Microsoft Azure [37], and IBM [38]. The detail resource characteristics which are used for modelling IaaS providers are shown in Table 2. The three different types of average VM **initiation time** are used in the experiment, and the mean initiation time are varied from 30 seconds to 15 minutes (standard deviation= (1/2)xmean). The mean of initiation time is calculated by conducting real experiments of 60 samples on GoGrid[35] and Amazon EC2[33] done for four days (2 week days and 2 weekend days). The VM initiation time is varied using normal distribution.

Overview of Performance Results

In this section, first, we 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 during we vary one parameter, the rest parameters are given constant mean vaule. The constant mean values, which are used during experiment, are as follows: arrival rate=5000 requests/sec, deadline=2*estprocT, budget=1 $, requst length= $4x10^6$MI, and penalty rate factor (r) =10.

Table 2. The summary of 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

A. Comparison with Reference Algorithms

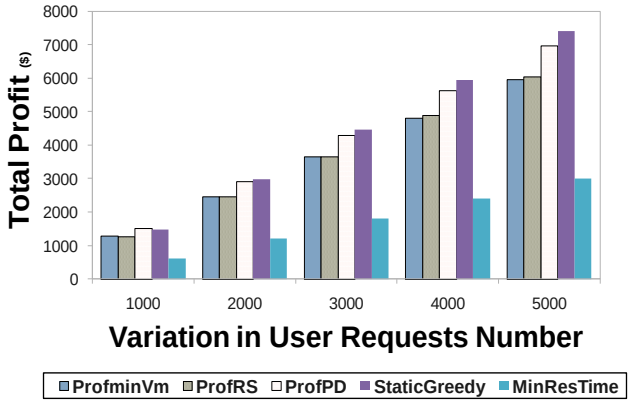
To observe the overall performance of our algorithms, we vary the number of users from 1000 to 5000 without varying all other factors such as deadline and budget. Figure 2 presents the comparison of our proposed algorithms with reference algorithms *StaticGreedy* and *MinResTime* in terms of four performance metrics. When the number of user requests varies from 1000 to 5000, for each algorithm the total profit and average response time has increased, because of the more user requests.

Figure 2 shows that *ProfPD* just generate 8% less profit (Requests = 5000) for SaaS provider than *StaticGreedy* which is used as the upper bound. That is because in the case of *StaticGreedy*, all the user requests are already known from the beginning to the SaaS provider. The base algorithm *MinResTime* even though results in the least (two third of *StaticGreedy*) response time, generate the least profit (approximately half of *ProfPD*). These observations indicate the trade-off between the response time and profit, which SaaS provider has to manage while scheduling the requests.

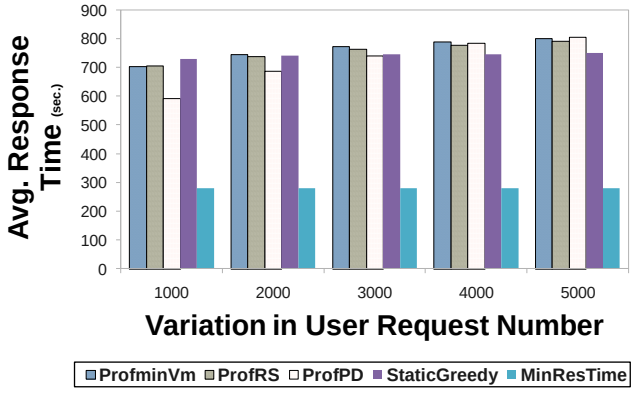
Figure 2a shows that the *ProfPD* achieves upto (15%) more profit over *ProfRS* and (17%) over *ProfminVM* by accepting (10%, 15%) more user requests and initiating (19%, 40%) less number of VMs, when number of users changes from 1000 to 5000. When the user number is 1000 *ProfPD* earn 4% and 15% more profit over

ProfminVM and *ProfRS* respectively. When the user number is increased from 1000 to 5000, the profit difference between *ProfPD* and other two algorithms has become larger. This is because when the number of requests increased, the number of users being accepted increased by utilizing initiated VMs. In such a way, the cost has been reduced to speed the profit increase. If all requests are known before scheduling, then *StaticGreedy* is the best choice for maximizing profit, however, in real Cloud computing market, these are unknown. Therefore, a SaaS provider should use *ProfPD*, however, *ProfRS* is a better choice for a SaaS provider in comparison with *ProfminVM*. In addition, we are able to demonstrate that *ProfPD* is effective in maximizing profit in heavy workload situations.

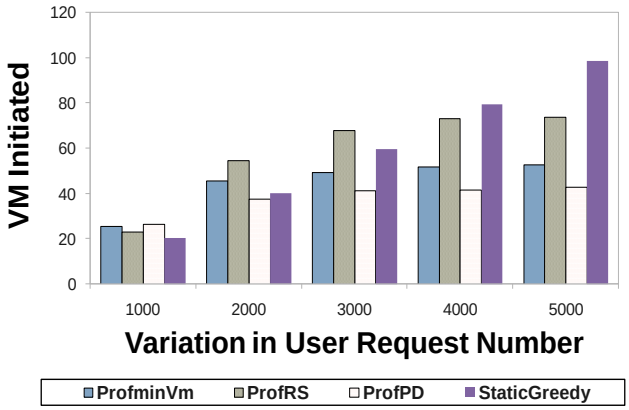
Figure 2b presents that all of our algorithms' trend of response time are increased from 1000 users to 5000 users because of increase processing of user requests per VM. When there is smaller number of users, the difference between different algorithm's response times becomes significant, for example, with 1000 users request, *ProfPD* gives users 16% lower response time than *ProfminVM* and *ProfRS*, and even accept more requests. This is because *ProfPD* scheduled less number of users per VM, thus user's experience less delay. In other scenarios the reason for lower response time is due to less initiation time. *ProfminVM* provides the lowest response time compare with others, because it can serve a new user with new VMs.



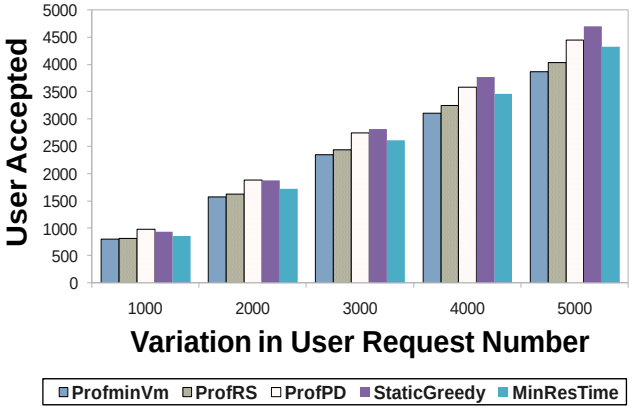
(a). Total profit



(b). Average response time



(c). Number of initiated VMs



(d). Number of accepted users

Figure 2. Overall algorithms' performance during variation in user numbers

B. Impact of QoS parameters

In the following sections, we examine various experiments by varying both user and resource provider side's SLA properties to analyse the impact of each parameter.

*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 such as deadline, budget as the same. All experiments are conducted with 5000 user requests. It can be seen from Figure 3, when arrival rate is "very high", the performance of *ProfminVM*, *ProfRS*, and *ProfPD* are affected significantly. The overall trend of profit is decreasing and the response time is increasing because when there is more users come per second, the service capability is decreased due to less new VM be initiated.

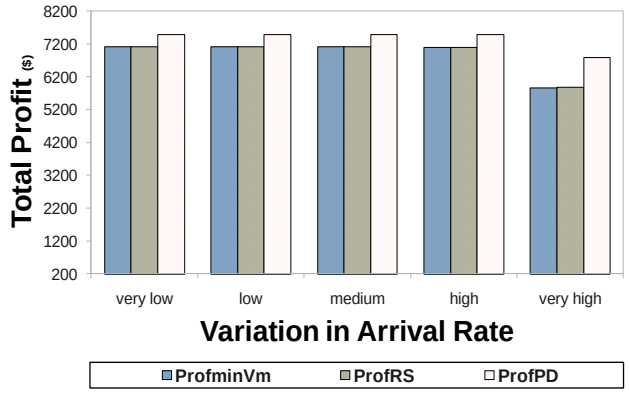
Figure 3a shows that the *ProfPD* achieves the highest profit (maximum 15% more than *ProfminVM* and *ProfRS*) by accepting (45%) more users and initiating the least number of VMs (19% less than *ProfminVM*, 28% less than *ProfRS*) when arrival rate is increased from "very small" to "very large". This is because *ProfPD* accept users with existing machines with penalty delay. In the same scenario, *ProfminVM* and *ProfRS* gain similar profit, but *ProfRS* accepts 4% more number of users with 13% more number of VMs than *ProfminVM*. Therefore, in this variation scenario *ProfPD* is the best choice for a SaaS provider, however, when arrival rate is "very large", and the number of VM is limited, *ProfRS* is a better choice compare with *ProfminVM* because although it provides similar profit as *ProfminVM*, yet it accepts more number of user requests, leading to market share expanding.

Figure 3b presents that the *ProfPD* results in the least response time and accepted more number of users with less number of VMs except when arrival rate is very high. Even in the case of high arrival rate, the difference between response time from *ProfPD* and its next competitor is just 3%. *ProfminVM* and *ProfRS* have similar response time. However, there is drastic jump in response time when arrival rate is very high because more number of users is accepted per VM which delays the processing of requests. It is safe to conclude that even considering the response time constraints from users, the first choice for a SaaS provider is still the *ProfPD*.

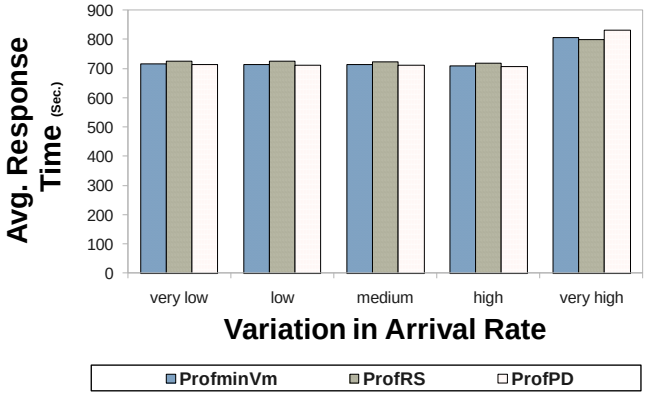
*2) Impact of **deadline** variation*

To investigate the impact of deadline in our algorithms, we vary the deadline, while keeping all other factors such as arrival rate and budget fixed. Figure 4a presents that the *ProfPD* gained the highest profit (45% over

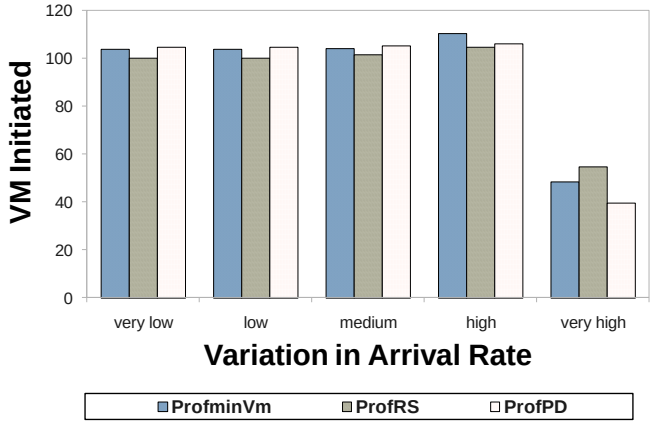
ProfminVM and 41% over *ProfRS*) by accepting 33% more user requests (Figure 4d) and initiating 52% less number of VMs (Figure 4c)". In some scenarios, *ProfminVM* provides higher profit than *ProfRS,* for example, when deadline is "very tight", because *ProfRS* accepted requests with larger service time, which occupy the space for accepting other requests. Hence, in general a SaaS provider should use *ProfPD* for maximizing profit.



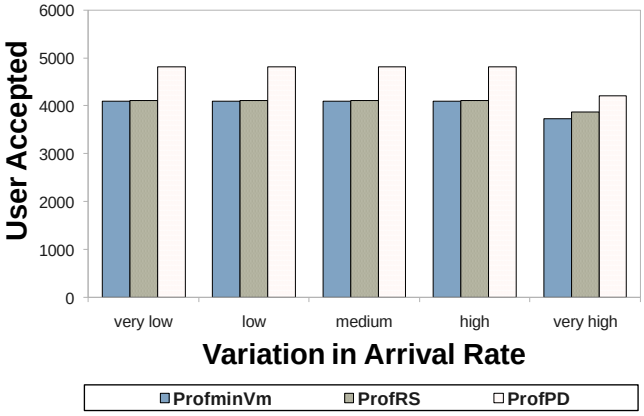
(a). Total profit



(b). Average response time



(c). Number of initiated VMs



(d). Number of accepted users

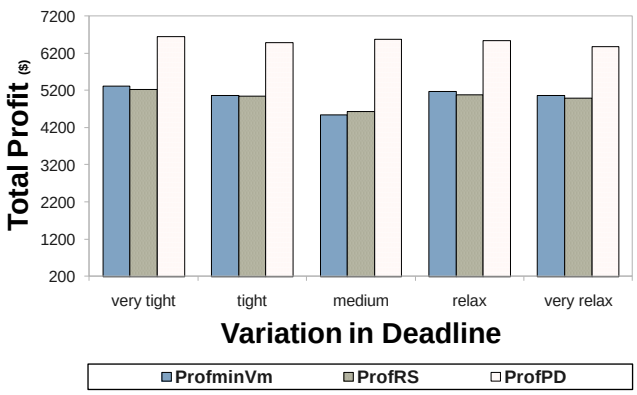
Figure 3. Impact of arrival rate variation

Figure 4b presents that when deadline is relaxed, *ProfPD* results 4% higher average response time than in the case of *ProfminVM* and *ProfRS*. The *ProfPD* responses the slowest to requests because of the two factors governing response time, i.e., request's service time and VM initiation time. It can be seen from Figure 4d, *ProfPD* always requires less number of VMs, to process more requests. Thus, when service time is comparable to the VM initiation time, the response time will be lower. When the VM initiation time is larger than the service time, the response time will be affected by the number of initiated VMs.

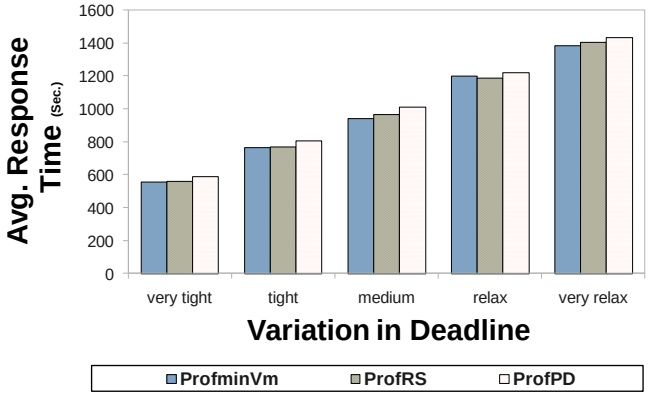
*3) Impact of **budget** variation*

Figure 5 shows how budget variation impact our algorithms, while keeping all other factors such as arrival rate,

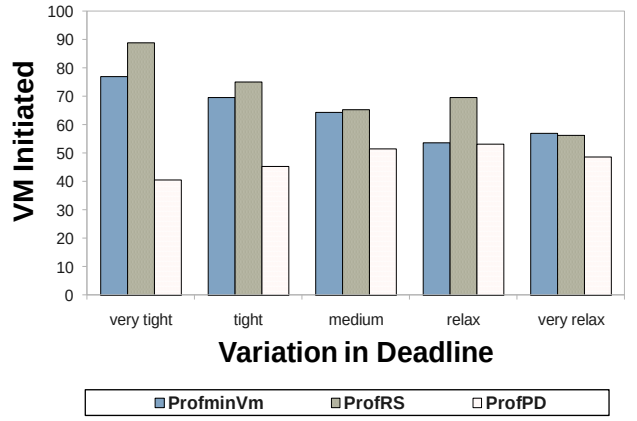
deadline as the same. Figure 5a shows that when budget is varied from "very small" to "very large", in average the total profit by all the algorithms has increased, and response time has decreased since the less number of requests are processed using more number of VMs. From Figure 5a, it can be observed that *ProfPD* gains the highest profit for SaaS provider except when budget is "large". In case of scenario when budget is "large",



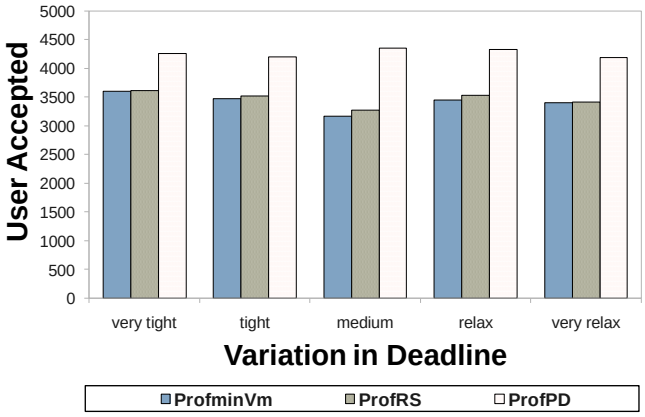
(a). Total profit



(b). Average response time



(c). Number of initiated VMs



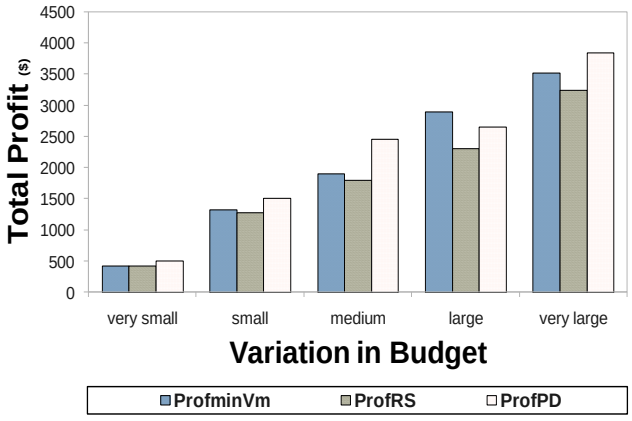
(d). Number of accepted users

Figure 4. Impact of deadline variation

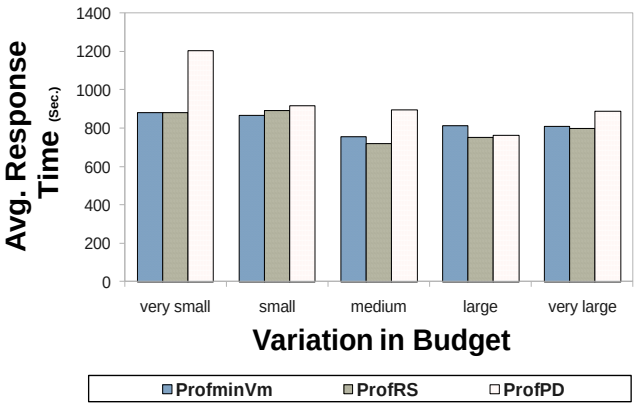
ProfminVM provides highest profit (20%) over others by accepting similar number of user requests with initiating more VMs without penalty delay. This is due to the increase in *Penalty Delay Rate (β)* (Equation 16) with the budget raise. Between *ProfminVM* and *ProfRS*, *ProfminVM* provides more profit in all scenarios. Therefore, in general a SaaS provider should consider *ProfPD*, *ProfminVM* is a better choice for a SaaS provider in comparison with *ProfRS*.

In the case of response time (Figure 5b), *ProfPD* on average delayed the processing of request longest (e.g. 33% more response time for "very small" budget scenario) even though it processed more user requests and initiated less VMs. However, when budget is "large", the response time provided by *ProfminVm* is the longest even though it accepts similar number of users as *ProfPD*. This anomaly is due to the contribution of VM initiation time which

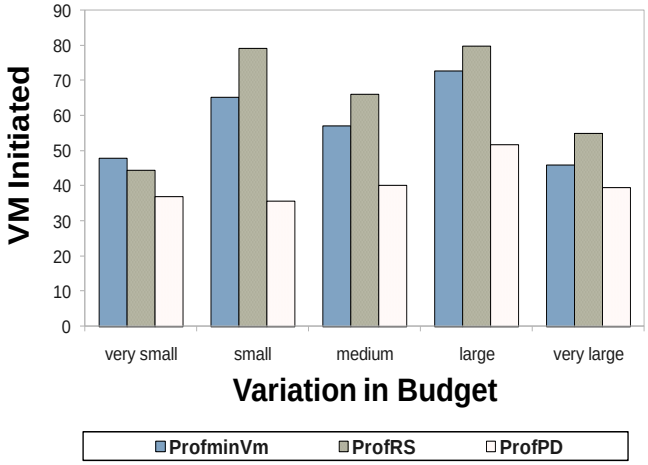
becomes very significant when *ProfRS* initiated large number of VM.



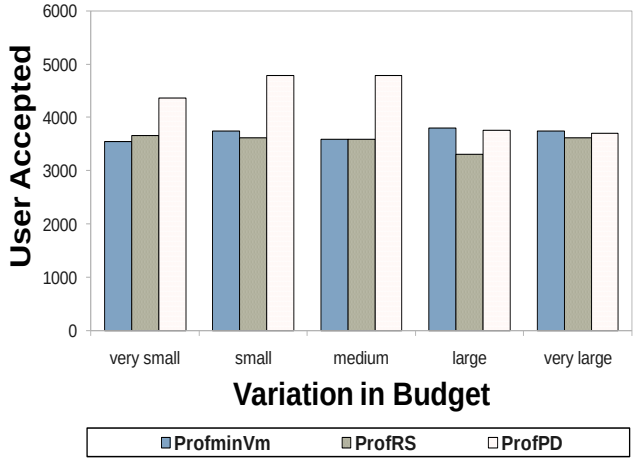
(a). Total profit



(b). Average response time



(c). Number of initiated VMs



(d). Number of accepted users

Figure 5. Impact of budget variation

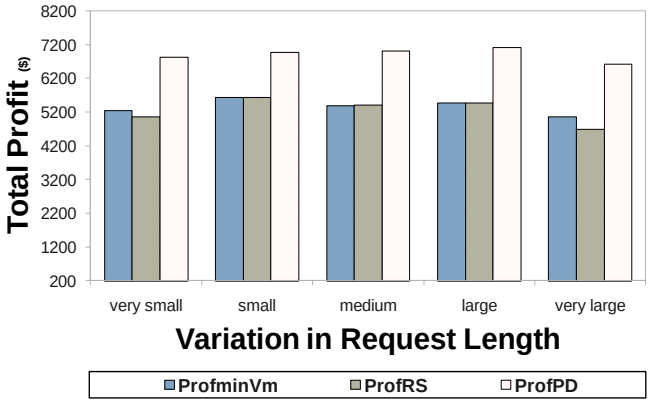
*4)Impact of **service time** variation*

Figure 6 shows how service time impact our algorithms, while keeping all other factors such as arrival rate, deadline as the same. In order to vary the service time, five classes of request length (*MI*) are chosen from "very small" (10^6MI) "very large" ($5x10^6$MI).

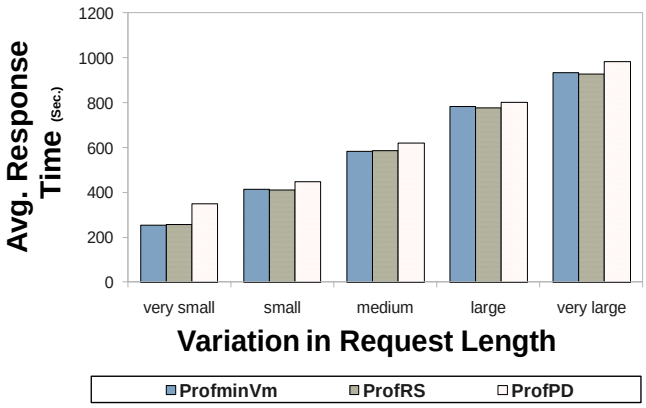
Figure 6a shows that the total profit by all algorithms has slightly decreased but response time increased rapidly when the request length is varied from "very small" to "very large". *ProfPD* achieves the highest profit among other competitor algorithms. For example, in the case of "very large" request length scenario, *ProfPD* generated about 30% more profit than other algorithms by accepting 24% more number of user requests (Figure 6d) and

initiating 32% (Figure 6c) less number of VMs. In addition, *ProfminVM* and *ProfRS* gain similar profit in most of the cases. Therefore, the *ProfPD* is the best solution for any size of requests.

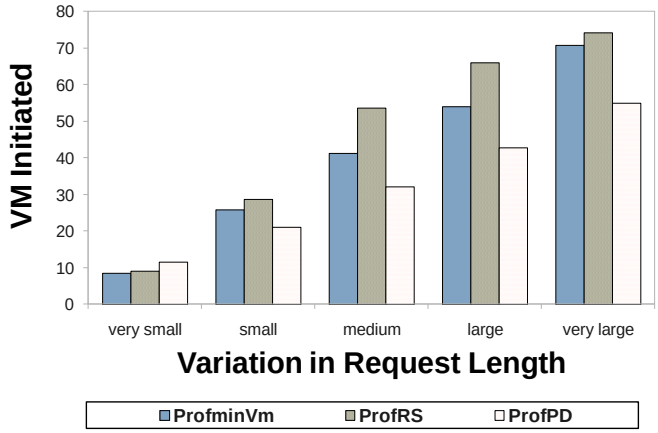
In addition, it can be observed from Figure 6b that *ProfPD* provides only a slightly higher response time (almost 6%) than others except when the request size is very small. When request size is very small, the response time provided by *ProfPD* becomes 27% more than others, this is because it accepts 63% more number of user requests with 22% more number of VMs, leading to the more user requests waiting longer for processing on each VM.



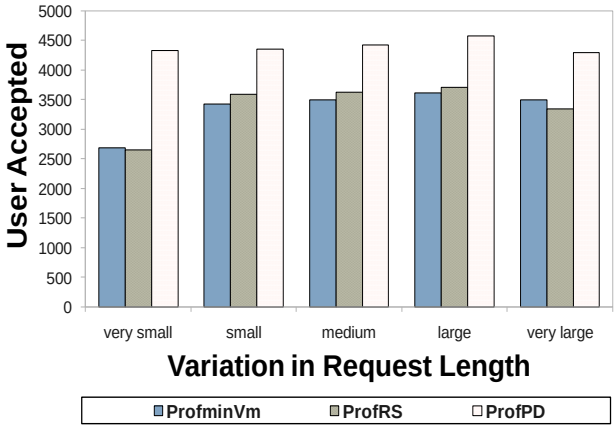
(a). Total profit



(b). Average response time



(c). Number of initiated VMs



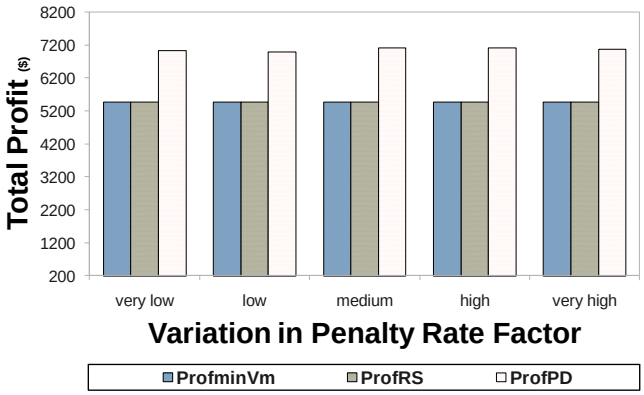
(d). Number of accepted users

Figure 6. Impact of request length variation

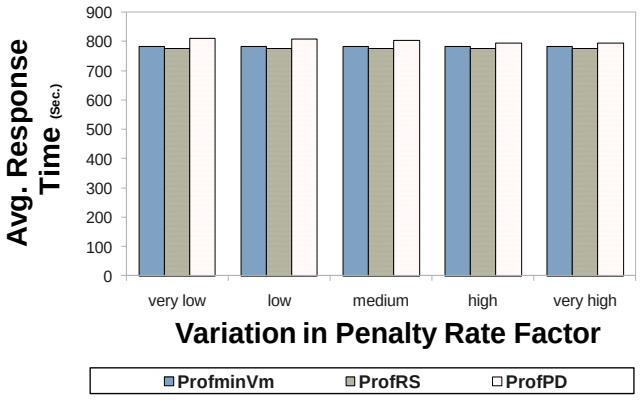
5)Impact of ***penalty rate*** varation

In this section, we investigate how penalty rate (β) impacts our algorithms. The penalty rate (Equation (16)) depends on how long user is willing to wait (r), which is defined as ***penalty rate factor*** in our paper. Therefore, when the penalty rate factor (r) is large, the penalty rate is small. All the results are presented in Figure 7.

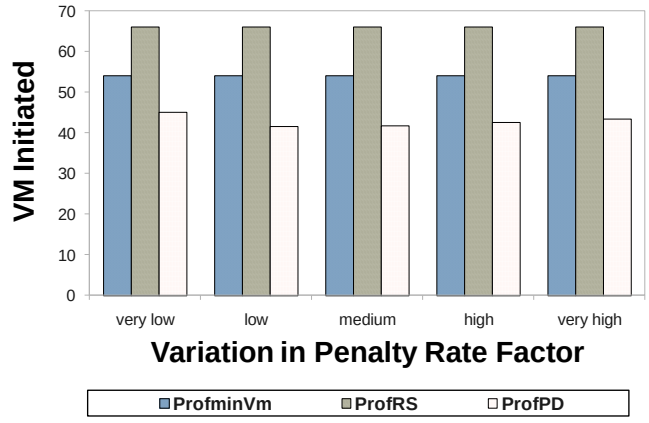
In can be observed from Figure 7 that only *ProfPD* shows some effect of variation in penalty rate since this is the only algorithm which uses Penalty Delay strategy to maximize the total profit. Even the total profit (Figure 7a) and average response time (Figure 7b) are only slightly decreased when the (*r*) is varied from "very low" to "very high". In almost all scenarios, *ProfPD* achieves 29% more profit over others by accepting 22% more number of users and initiating 30% less number of VMs. In addition, when the penalty rate varies from "very low" to very high", the response time slightly decreased. This is because *ProfPD* accept a little bit less number of users with similar number of VMs. Thus, the number of user requests waiting in each VM becomes less, leading to faster response time for each request.



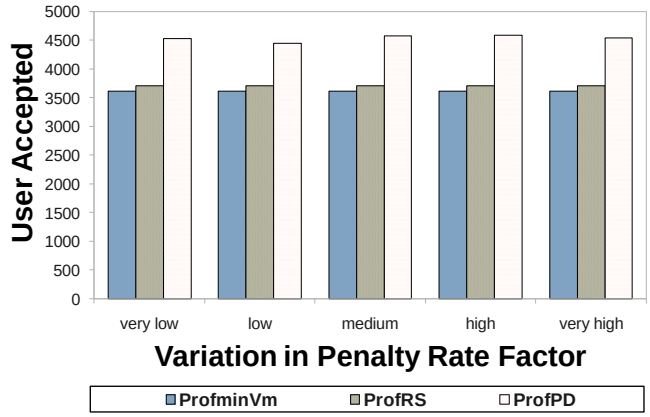
(a). Total profit



(b). Average response time



(c). Number of initiated VMs



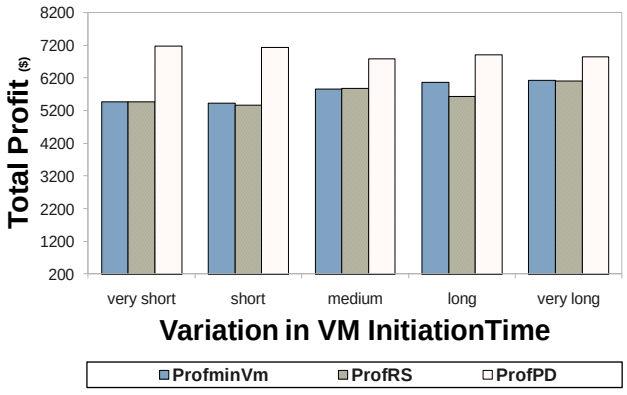
(d). Number of accepted users

Figure 7. Impact of penalty rate factor variation

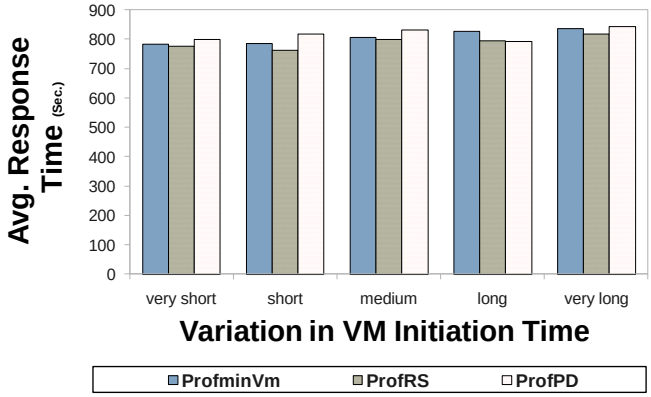
6)Impact of ***Initiation Time*** variation

In this section, we analyse how our algorithms perform for different initiation times which are varied from "low" to "high". Figure 8a illustrates that with increase in initiation time the total profit gained from all the algorithms decreases slightly while response time has increased a little bit. Due to increase in initiation time, the number of initiated VMs (Figure 8c) has decreased rapidly due to the contribution of initiation time in SaaS providers cost (spending). In all the scenarios, still *ProfPD* gains highest profit over others by accepting almost upto 17% more user requests (Figure 8d) and with almost upto 37% less initiated VMs. Therefore, *ProfPD* is the best choice for a SaaS provider in all scenarios.

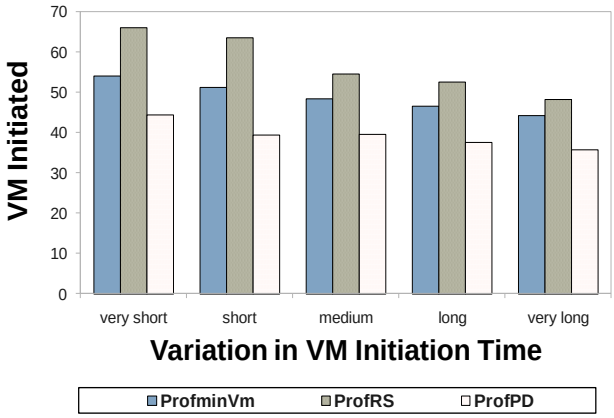
The response time offered by *ProfPD* is slightly higher than others in most of cases, because it accepted more users with less number of VMs, in other word, a VM required to serve more number of users, leading to delay in request processing. The anomaly is when initiation time is "long". The response time of *ProfPD* is the lowest in this scenario, because due to large initiation time of VM, the response time is also increased with each initiated VM. However, the contribution to delay in processing of requests, due to more number of requests per VM, will also increase. This leads to higher response time in the scenario when the initiation time is "very long".



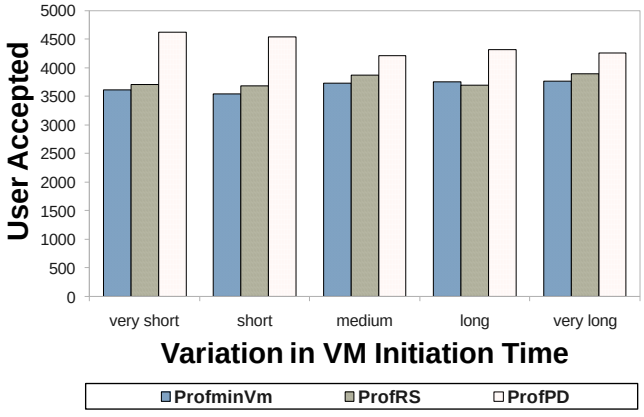
(a). Total profit



(b). Average response time



(c). Number of initiated VMs



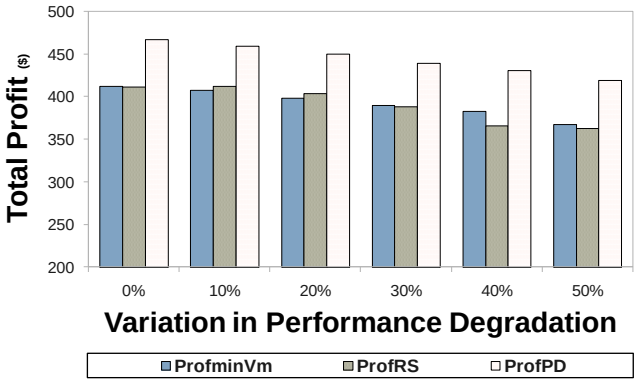
(d). Number of accepted users

Figure 8. Impact of initiation time variation

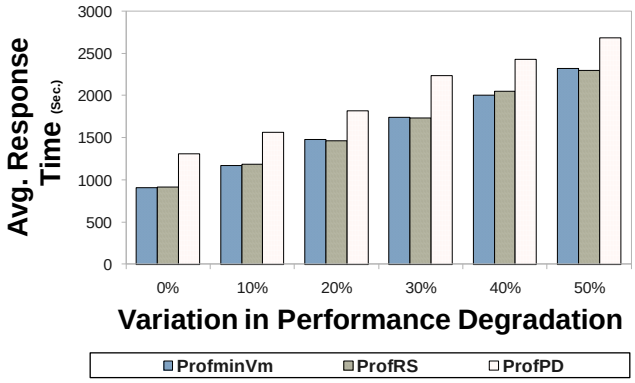
C. Robustness Analysis

In order to evaluate the robustness of our algorithms, we run some experiments by reducing the actual performance of VMs in the SLA(R) promised by IaaS providers. This performance degradation has been observed by previous research study in Cloud computing environments [39]. This experiment is conducted also to signify the inclusion of compensation (penalty) clauses in SLAs which is absent in current IaaS providers' SLAs [34]. We modelled the reduced performance using normal distribution, and mean varies from 0% to 50% equally.

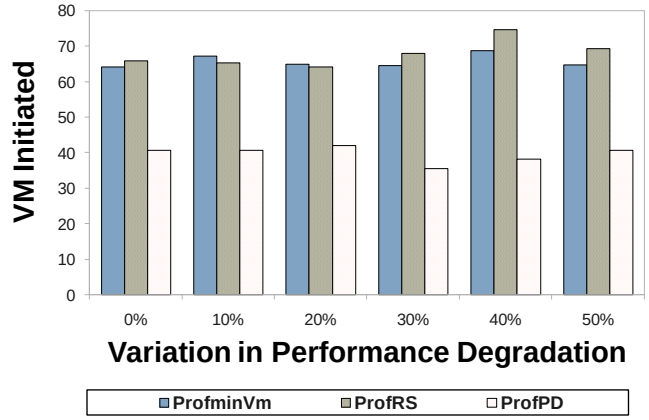
Figure 10 shows that during the degradation of VM performance, the average total profit (Figure 10a) has reduced 11% and average response time (Figure 10b) has doubled with the increase in performance degradation of initiated VMs. This is because of the performance degradation of VMs has not been accounted in SLA(R). Therefore, a SaaS provider does not consider this variation during their scheduling, but it impacts significantly on the total profit and average user requests response time.

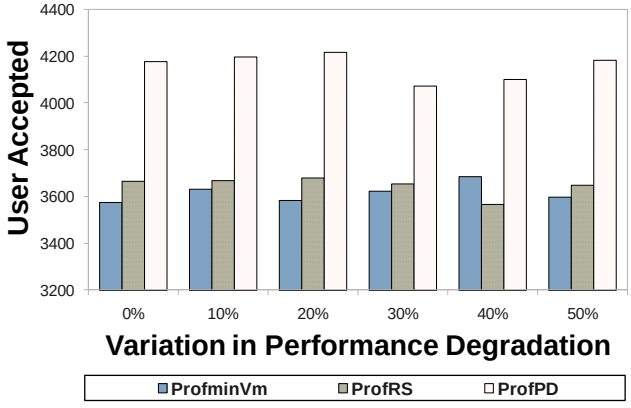


(a). Total profit



(b). Average response time

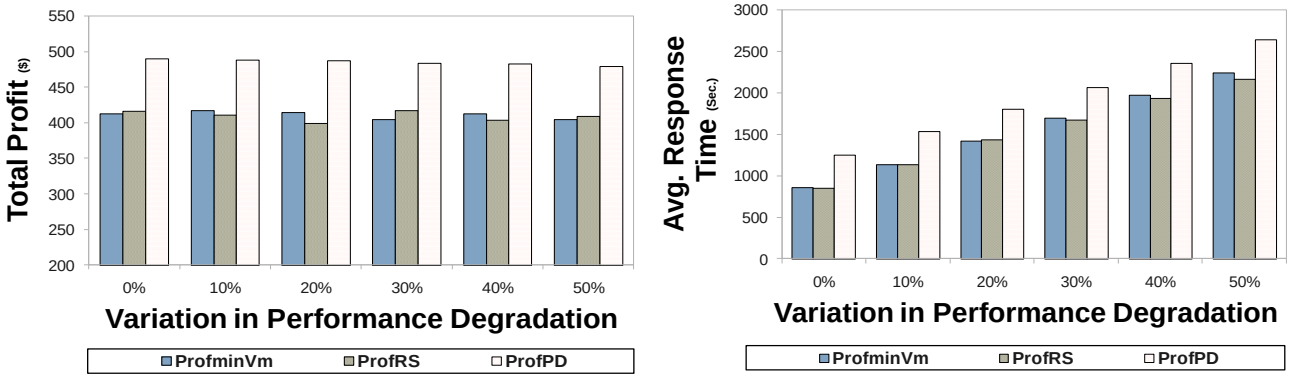




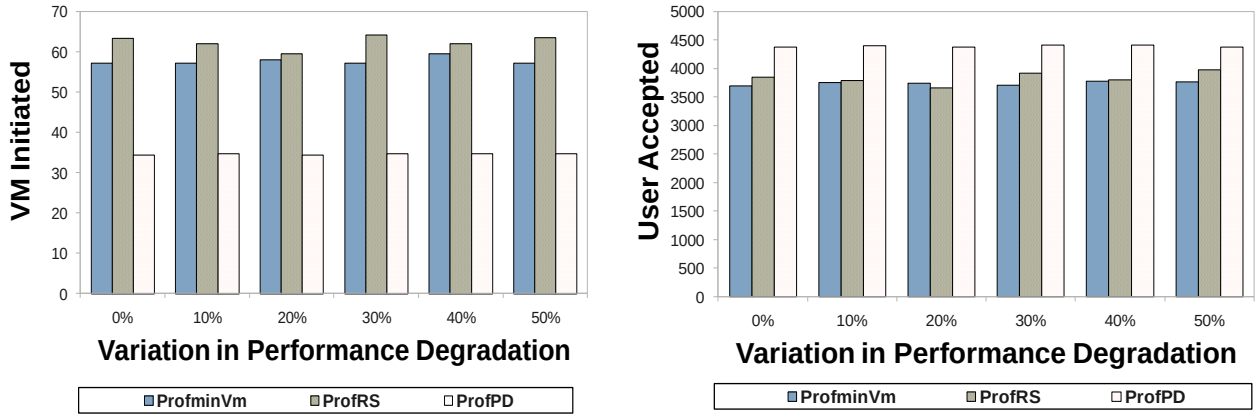
(c). Number of initiated VMs (d). Number of accepted users

Figure 10. Impact of performance degradation variation

Two solutions to handle this VMs performance degradation are: firstly, use the penalty clause in SLA(R) to compensate for profit loss; secondly, consider the degradation as a potential risk. Therefore, while scheduling we add a (300 seconds) slack time in estimated service request processing time and it can be seen from Figure 11, that the impact on the total profit due to degradation has reduced considerably (from 0% to 50%, profit decreased only by 2%). Thus, if there is a risk for a SaaS provider to enforce SLA violation with an IaaS provider, an alternative solution to reduce risk is by considering a slack time during scheduling.



(a). Total profit (b). Average response time



(c). Number of initiated VMs (d). Number of accepted users

Figure 11. Impact of performance degradation variation after considering slack time

VI. CONCLUSIONS AND FUTURE DIRECTIONS

In Cloud computing environments, primarily three types of on-demand services are available for users i.e. Software as a Service, Infrastructure as a Service and Platform as a Service. This paper focuses on admission control and scheduling user requests for SaaS providers with the explicit aim of profit maximization without violating SLAs signed with users. Thus, to achieve this goal, we presented three profit driven algorithms which consider various QoS parameters from both users' and providers' side such as deadline, budget, penalty rate, service time, and VM initiation time. Simulation results show that in average the *ProfPD* algorithm gives the maximum profit among all proposed algorithms by varying parameter. If a user request needs fast response time, *ProfRS* and *ProfminVM* could be chosen depending on the scenario. The summary of algorithms and their ability to deal with different scenarios is shown in Table 3.

In this work we assumed that the estimated service will be accurate since one can use existing performance estimation techniques (e.g. analytical modeling [23], empirical, and historical data [24]) to predict very accurate service times on various types of VMs. But still some errors exist in this estimated service time [39] due to VMs' performance degradation in Cloud computing environment. This error could be solved by two strategies: firstly, consider the penalty compensation clause in SLAs with IaaS provider and enforce SLA violation; secondly, during scheduling add some slack time for preventing risk.

Table 3. Summary of heuristics of comparison results (Profit)

Algorithm	**Time complexity**	**Overall Performance**						
		Arrival Rate	Deadline	Budget	Request Length	Penalty Rate Factor	VM Initiation Time	Data Transfer
ProfminVM	*O(KJ+K)*	Good (low - high)	Good (low-high)	Good	Good (very low & very high)	No effect	Okay	Good (very low & very high)
ProfRS	$O(KJ+K^2)$	Okay (very	Okay (very	Okay (very	Okay	No effect	Good (low-	Okay

		high)	high)	low)			high)	
ProfPD	$O(KJ+K^2)$	Best	Best	Best	Best	Best	Best	Best

Although we have added some slack time, in the future we will try to increase the robustness of our algorithms by handling such errors dynamically. In addition, due to this performance degradation error, we would like to consider SLA negotiation in Cloud computing environment to improve the robustness. We also like to 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 maximize a SaaS provider's profit will be one of our future direction for improving our algorithms' time complexity.

REFERENCES

[1] Yeo, C.S., and Buyya, R. (2005). 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] Jaideep, D.N., and Varma, M.V. (2010). Learning based Opportunistic admission control algorithms for map reduce as a service. In Procedings of the 3rd India Software Engineering Conference (ISEC 2010), Mysore, India.

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

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

[5] Irwin, D.E. and Grit, L.E. and Chase, J.S. (2004) 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.

[6] Yemini, Y. (1981). 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.

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

[8] Buyya, R., Yeo, C. S., Venugopal, S, Broberg, J and Brandic, I. (2009). 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.

[9] Vaquero, L.M., Rodero-Merino, L., Caceres, J., and Lindner, M. (2009). A break in the clouds: towards a cloud definition, ACM SIGCOMM Computer Communication Review, 39(1), (pp.50-55).

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

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

[12] Youseff, L., Butrico, M. and Silva, D. (2008). Toward a Unified Ontology of Cloud Computing. In Proceedings of 2008 Grid Computing Environments Workshop (GCE 2008), Austin, Texas, USA.

[13] Bichler, M. and Setzer, T. (2007). Admission control for media on demand services. Service Oriented Computing and Application. In Proceedings of IEEE International Conference on Service Oriented Computing and Applications (SOCA 2007), Newport Beach, California, USA.

[14] Chun, N. B. and Culler, D.E. (2002). User-centric performance analysis of market-based cluster batch schedulers. In Proceedings of the 2nd IEEE/ACM International Symposium on Cluster and Grid Computing (CCGrid 2002), Berlin, Germany.

[15] Coleman, K., Norris, J., Candea, G. and Fox, A. (2004). OnCall: Defeating spikes with a free-market application cluster, In Proceedings of the 1st International Confererance on Autonomic Computing. NY, US.

[16] Broberg, J., Venugopal, S., and Buyya, R. (2008). Market-oriented Grids and Utility Computing: The state-of-the-art and future directions, Journal of Grid Computing, 3(6), (pp.255-276).

[17] Buyya, R., Ranjan R., and Calheiros, R. N. (2010). InterCloud: Utility-Oriented Federation of Cloud Computing Environments for Scaling of Application Services, Proceedings of the 10th International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP 2010), Busan, South Korea.

[18] Rochwerger, B. et al. (2009). The Reservoir Model and Arhitecture for Open Federated Cloud Computing. IBM Systems Jaunal, 4 (53), (pp.1-11).

[19] Keahey, K., Matsunaga, A., and Fortes, J. (2009). Sky computing, IEEE Internet Computing, 13(5), (pp. 43–51).

[20] Buyya, R., Ranjan, R. and Calheiros, R.N. (2009). Modeling and Simulation of Scalable CloudComputing Environments and the CloudSim Toolkit: Challenges and Opportunities. In Proceedings of the 7th High Performance Computing and Simulation Conference, Leipzig, Germany.

[21] Nudd, G.R., Kerbyson, D.J., Papaefstathiou, E., Perry, S.C., Harper, J.S., and Wilcox, D.V. (2000). Pace-A Toolset for the Performance Prediction of Parallel and Distributed Systems. International Journal of High Performance Computing Applications, 14(3), (pp. 228–251).

[22] Smith, W. and Foster, I. and Taylor, V. (1998). Predicting Application Run Times Using Hitorical Inforamtion. In Proceedings of IPPS/SPDP Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP 1998), Florida, USA.

[23] Animoto, retrieved on 12nd Sep 2010. http://developer.amazonwebservices.com/connect/entry!default.jspa;jsessionid=5E63CF198EA2949E920D500303C858CE?categoryID=89&externalID=932&fromSearchPage=true

[24] Liu, Z., Squillante, M.S. and Wolf, J.L. (2001). On Maximizing Service-Level-Agreement Profits. In Proceedings of the 3rd ACM conference on Electronic Commerce (EC 01), Tampa, Florida, USA

[25] Menasce D. A., Almeida V. A. F., Fonseca R., and Mendes M. A. (1999). A methodology for workload characterization of e-commerce sites. In Procedings of the 1999 ACM Conference on Electronic Commerce (EC 1999), Denver, CO, USA.

[26] Chen, Y., Iyer, S., Liu, X., Milojicic, D., and Sahai, A. (2007). SLA Decomposition: Traslating Service Level Objectives to System Level Thresholds, In Proceedings of 4th IEEE Internaltional Conference on Autonomic Computing, Florida, USA.

[27] Reig, G. Alonso, J. and Guitart, J. (2010). 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.

[28] Xiong, K., Perros, H. (2008). SLA-based Resource Allocation in Cluster Computing Systems. In Proceedings of 17th IEEE International Symposium on Parallel and Distributed Processing (IPDPS 2008), Alaska, USA.

[29] Netto, M. and Buyya, R. (2009). Offer-based Scheduling of Deadline-Constrained Bag-of-Tasks Applications for Utility Computing Systems, Proceedings of the 18th International Heterogeneity in Computing Workshop (HCW 2009), in conjunction with the 23rd IEEE International Parallel and Distributed Processing Symposium (IPDPS 2009), Roma, Italy.

[30] Garg, S. K., Buyya, R., and Siegel, H. J. (2009). Time and Cost Trade-off Management for Scheduling Parallel Applications on Utility Grids, Future Generation Computer Systems,26(8), (pp. 1344-1355).

[31] Islam, M., Balaji, P., Sadayappan, P. and Panda, D. K. QoPS: A QoS Based Scheme for Parallel Job Scheduling. In Proceedings of the 9th International Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP 2003), Seattle, USA.

[32] Islam, M., Sadayappan, P., and Panda, D. K. (2004). 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.

[33] Varia, J. (2010), Architecting Applications for the Amazon Cloud, Cloud Computing: Principles and Paradigms, R. Buyya, J. Broberg, A.Goscinski (eds), ISBN-13: 978-0470887998, Wiley Press, New York, USA. Web - http://aws.amazon.com

[34] CIO, retrieved on 10 Sep 2010: http://www.cio.com.au

[35] GoGorid, retrieved on 10 Sep 2010: http://www.gogrid.com

[36] RackSpace, retrieved on 10 Sep 2010: http://www.rackspacecloud.com

[37] Microsoft Azure, retrieved on 10 Sep 2010: http://www.microsoft.com/windowsazure/

[38] IBM, retrieved on 10 Sep 2010: http://www.ibm.com/ibm/cloud/ibm_cloud/

[39] Ostermann, S., Iosup, A., Yigitbasi, M.N., Prodan, R., Fahringer, T. and Epema, D. (2009). An early performance analysis of cloud computing services for scientific computing. In Proceedings of the 1st International Conference on Cloud Computing (Cloudcom 2009), Beijing, China.

[40] Dinesh, V. (2004). Supporting Service Level Agreements on IP Networks. In Proceedings of IEEE/IFIP Network Operations and Management Symposium, 92(9), (pp. 1382-1388), NY, USA.

[41] Kumar, S., Dutta, K., Mookeriee, V. (2009). Maximizing business value by optimal assignment of jobs to resources in grid computing, European Journal of Operational Research, 194(3).