Journal of Computer and System Sciences 78 (2012) 1280–1299

Contents lists available at SciVerse ScienceDirect

Journal of Computer and System Sciences

www.elsevier.com/locate/jcss

SLA-based admission control for a Software-as-a-Service provider in Cloud computing environments

Linlin Wu *, Saurabh Kumar Garg, Rajkumar Buyya

Cloud Computing and Distributed Systems (CLOUDS) Laboratory, Department of Computer Science and Software Engineering, The University of Melbourne, Australia

ARTICLE INFO

Article history:
Received 25 August 2010
Received in revised form 20 May 2011
Accepted 8 December 2011
Available online 23 December 2011

Keywords:
Cloud computing
Service Level Agreement (SLA)
Admission control
Software as a Service
Scalability of application services

ABSTRACT

Software as a Service (SaaS) provides access to applications to end users over the Internet without upfront investment in infrastructure and software. To serve their customers, SaaS providers utilise resources of internal data centres 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 utilise public Cloud resources to maximize profit by minimizing cost and improving customer satisfaction level. Furthermore, we conduct an extensive evaluation study to analyse which solution suits best in which scenario to maximize SaaS provider's profit. Simulation results show that our proposed algorithms provide substantial improvement (up to 40% cost saving) over reference ones across all ranges of variation in QoS parameters.

1. 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 [8,9,11]. Considering the best known Cloud service providers, such as Saleforce.com [39], Microsoft [33], and Amazon [29], 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 [39] hosts resources but Animoto [20] rents resources from Amazon EC2 [29]. 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 [40].

To overcome the above limitations, multiple IaaS providers and admission control are considered in this paper. Procuring from multiple IaaS providers brings huge amount of resources, various price schemas, and flexible resource performance

* Corresponding author.
E-mail addresses: linwu@csse.unimelb.edu.au (L. Wu), sgrag@csse.unimelb.edu.au (S. Kumar Garg), raj@csse.unimelb.edu.au (R. Buyya).

0022-0000/$ – see front matter
doi:10.1016/j.jcss.2011.12.014

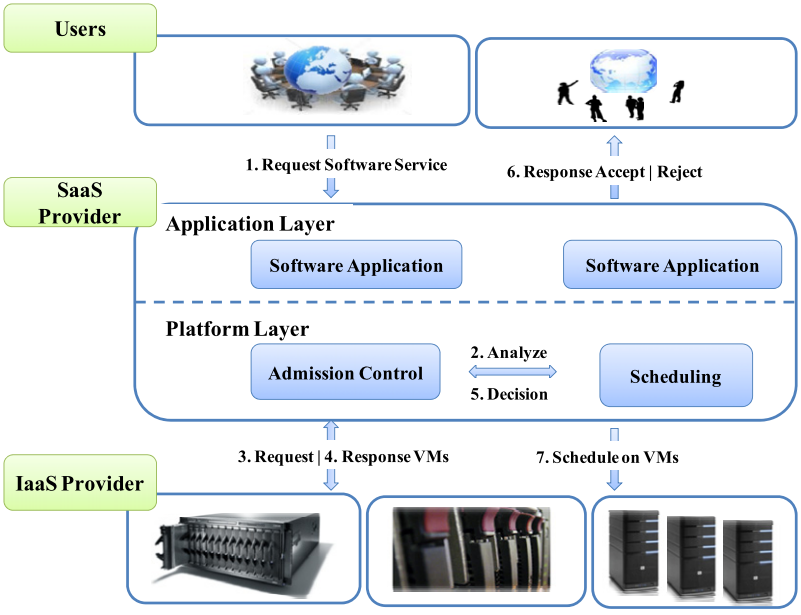


Fig. 1. A high level system model for application service scalability using multiple IaaS providers in Cloud.

to satisfy Service Level Objectives, which are items specified in Service Level Agreement (SLA). Admission control has been used as a general mechanism to avoid overloading of resources and SLA satisfaction [8]. However, current SaaS providers do not have admission control and how they conduct scheduling is not publicly known. Therefore, the following questions need to be answered to allow efficient use of resources in the context of multiple IaaS providers, where resources can be dynamically expanded and contracted on demand:

- Can a new request be accepted without impacting accepted requests?
- How to map various user requests with different QoS parameters to VMs?
- What available resource should the request be assigned to? Or should a new VM be initiated to support the new request?

This paper provides solutions to the above questions by proposing an innovative cost-effective admission control and scheduling algorithms to maximize the SaaS provider's profit. Our proposed solutions are able to maximize the number of accepted users through the efficient placement of request on VMs leased from multiple IaaS providers. We take into account various customer's QoS requirements and infrastructure heterogeneity. The key contributions of this paper are twofold: 1) we proposed system and mathematical models for SaaS providers to satisfy customers. 2) we proposed three innovative admission control and scheduling algorithms for profit maximization by minimizing cost and maximizing customer satisfaction level.

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. 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. Simulation results show that proposed algorithms improve the profit (up to 40% improvement) compared to reference algorithms by varying all range of QoS parameters. Prior related works are compared in Section 5. Finally, in Section 6, we conclude the paper by summarizing the comparison results and future work.

2. System model

In this section, we introduce a model of SaaS provider, which consists of actors and 'admission control and scheduling' system (as depicted in Fig. 1). 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. 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:

2.1.1. 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. 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 the SaaS provider misses 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?

2.1.2. 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 efficiently using resources from IaaS providers, and improving Customer Satisfaction Level (CSL) by satisfying SLAs, which are used to guarantee QoS requirements of accepted users. From SaaS provider's point of view, there are two layers of SLA with both users and resource providers, which are described in Section 2.1.1 and Section 2.1.3. It is important to establish two SLA layers, because SLA with user can help the SaaS provider to improve the customer satisfaction level by gaining users' trust of the quality of service; SLA with resource providers can enforce resource providers to deliver the satisfied service. If any party in the contract violates its terms, the defaulter has to pay for the penalty according to the clauses defined in the SLA.

2.1.3. IaaS provider

An IaaS provider (*RP*), 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. 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 [30]. The SLA(R) includes the following properties:

- **Service 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 (their own 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.

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 $inPri_j$ and $outPri_j$ 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* 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:

$$Prof_{ijl}^{new} = B^{new} - Cost_{ijl}^{new}; \quad \forall i \in I,\ j \in J,\ l \in N_j \tag{1}$$

The total cost incurred to SaaS provider for accepting the new request consists of request's processing cost (PC_{ijl}^{new}), data transfer cost (DTC_{jl}^{new}), VM initiation cost (IC_{ijl}^{new}), and penalty delay cost ($PDC_{ijl}^{new}T$) (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.

$$Cost_{ijl}^{new} = PC_{ijl}^{new} + DTC_{jl}^{new} + IC_{ijl}^{new} + PDC_{ij}^{new}; \quad \forall i \in I,\ j \in J,\ l \in N_j \tag{2}$$

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

$$PC_{ijl}^{new} = procT_{ijl}^{new} \times P_{jl}, \quad \forall i \in I,\ j \in J,\ l \in N_j \tag{3}$$

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

$$DTC_{jl}^{new} = inDS^{new} \times inPri_{jl} + outDS^{new} \times outPri_{jl}; \quad \forall j \in J,\ l \in N_j \tag{4}$$

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

$$IC_{ijl}^{new} = iniT_{ij} \times P_{jl}, \quad \forall i \in I,\ j \in J,\ l \in N_j \tag{5}$$

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

$$PDC_{ijl}^{new} = \beta^{new} \times PDT_{ijl}^{new}; \quad \forall i \in I,\ j \in J,\ l \in N_j \tag{6}$$

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 $\sum_{k=1}^{K} procT_{ijl}^{k}$, where K is the number of request yet to be processed before the new request. Thus, PDT_{ljl}^{new} is given by:

$$PDT_{ijl}^{new} = \begin{cases} t + \sum_{k=1}^{K} procT_{ijl}^{k} + procT_{ijl}^{new} - DL^{new}, & \text{if new VM is not initiated} \\ procT_{ijl}^{new} + iniT_{ijl} + DTT_{ijl}^{new} - DL^{new}, & \text{if new VM is initiated} \end{cases} \tag{7}$$

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

$$DTT_{ijl}^{new} = inDT_{ijl}^{new} + outDT_{ijl}^{new}; \quad \forall i \in I,\ j \in J,\ l \in N_j \tag{8}$$

Thus, the response time (T_{ijl}^{new}) for the new request to be processed on VM_{il} of IaaS provider j is calculated in Eq. (9) and consists of VM initiation time ($iniT_{ijl}^{new}$), request's service processing time ($procT_{ijl}^{new}$), data transfer time (DTT_{ijl}^{new}), and penalty delay time (PDT_{ijl}^{new}).

$$T_{ijl}^{new} = \begin{cases} \sum_{k=1}^{K} procT_{ijl}^{k} + procT_{ijl}^{new}, & \text{if new VM is not initiated} \\ procT_{ijl}^{new} + iniT_{ijl} + DTT_{ijl}^{new}, & \text{if new VM is initiated} \end{cases} \tag{9}$$

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

$$ret_{ijl}^{new} = \frac{prof_{ijl}^{new}}{T_{ijl}^{new}}; \quad \forall i \in I,\ j \in J,\ l \in N_j \tag{10}$$

3. Algorithms and strategies

In this section, we present four strategies to analyse whether a new request can be accepted or not based on the QoS requirements and resource capability. 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. 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**. Inputs of all strategies are QoS parameters of the new request and resource providers' related information. Outputs of all strategies are admission control and scheduling related information, for example, which VM and in which resource provider the request can be scheduled. All flow charts in this section are in the context of each VM in each resource provider.

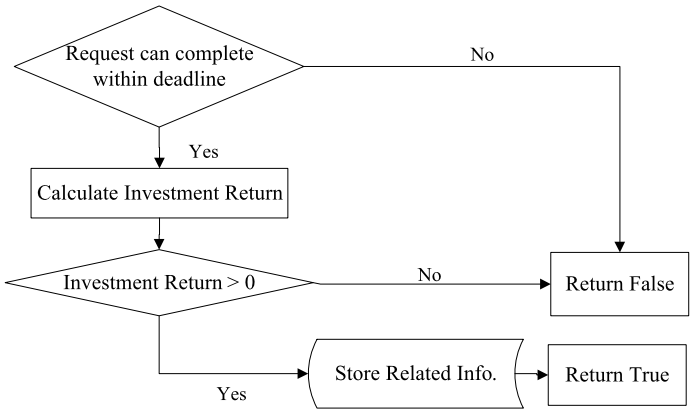


Fig. 2. Flow chart of 'initiate new VM strategy'.

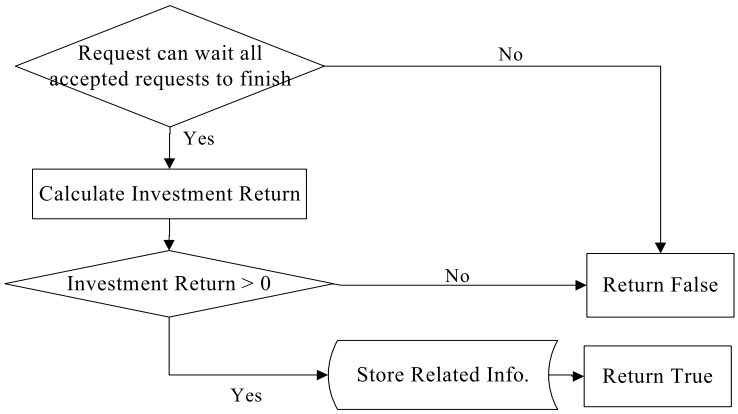


Fig. 3. Flow chart of 'wait strategy'.

3.1.1. Initiate new VM strategy

Fig. 2 illustrates the flow chart of "*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. This strategy is represented as *canInitiateNewVM*() in algorithms.

3.1.2. Wait strategy

Fig. 3 illustrates the "*wait strategy*", which first verifies each VM in each resource provider if the flexible time (fT_{ijl}^{new}) of the new request is enough to wait all accepted requests in vm_{il} to complete. The fT_{ijl}^{new} is given by Eq. (11), in which K indicates total number of all accepted requests, I indicates all VMs, J indicates all resource providers, l indicates VM type, and N_j indicates all VM types provided by resource provider j.

$$fT_{ijl}^{new} = DL^{new} - \sum_{k=1}^{K} procT_{ijl}^{k} - subT^{new}; \quad \forall i \in I,\ j \in J,\ k \in K,\ l \in N_j \tag{11}$$

If new request can wait for all accepted requests to complete, and then the investment return is calculated and the remaining steps are the same as those in initiate new VM strategy. This strategy is called as *canWait*() in algorithms.

3.1.3. Insert strategy

Fig. 4 shows the flow chart of "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 (fT_{ijl}^{k}) is enough to wait for a new user request to complete then the new request is inserted before request k. The fT_{ijl}^{k} indicates the duration of request

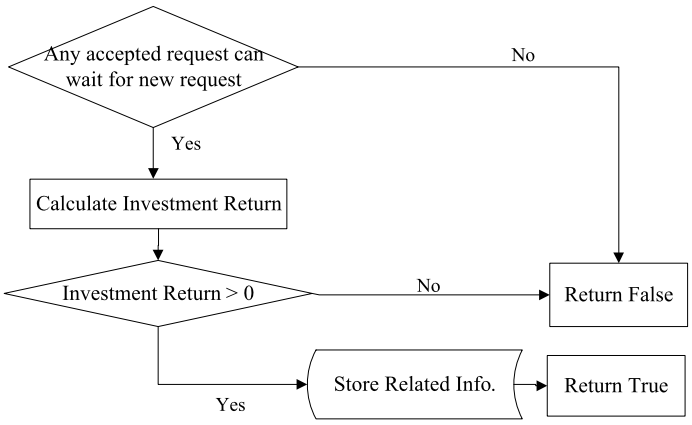


Fig. 4. Flow chart of 'insert strategy'

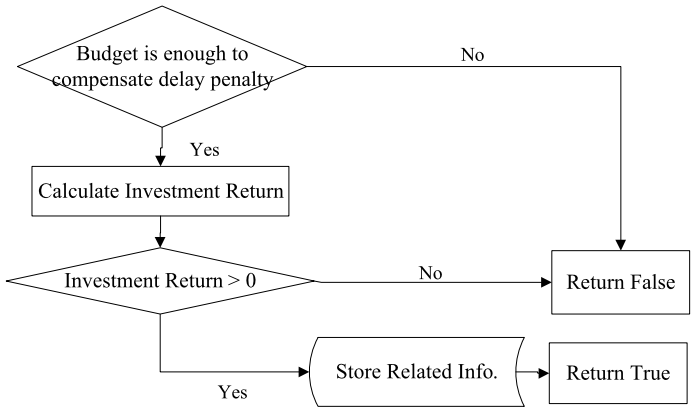


Fig. 5. Flow chart of 'penalty delay strategy'.

wait time with deadline and it is given by Eq. (12), in which DL^k indicates the deadline of accepted request, k indicates the position of accepted request, and K indicates the total number of accepted user requests, l indicates the VM type and N_j indicates all VM types provided by resource provider j.

$$fT_{ijl}^{k} = DL^{k} - \sum_{\substack{n=1,\\ n\neq k}}^{K} procT_{ijl}^{n} - T_{ijl}^{new} - subT^{new}; \quad \forall i \in I,\ j \in J,\ k \in K,\ l \in N_j \tag{12}$$

If there is an already accepted request u^k that is able to wait for the new user request to complete, the strategy checks if the new request can complete before its deadline. If so, u^{new} gets priority over u^k, then the algorithm calculates the investment return and the remaining steps are the same as those in *initiate new VM strategy*. This strategy is presented as *canInsert*() in algorithms.

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

A 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*:

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

3.2.1. Maximizing the profit by minimizing the number of VMs (ProfminVM)

Algorithm 1 describes the *ProfminVM* algorithm, which involves two main phases: a) **admission control** and b) **scheduling**.

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. 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* (Step 3). 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* (Step 8). 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 (Step 9). Otherwise, the algorithm gets the maximum investment return from all of the possible solutions (Step 13). The decision also depends on the minimum expected investment return ($expInvRet_{ijl}^{new}$) of the SaaS provider. If the investment return ret_{ijl}^{new} is more than the SaaS provider's $expInvRet_{ijl}^{new}$, the algorithm *accepts* the new request (Steps 14, 15), otherwise it *rejects* the request (Steps 16, 17). The expected investment return ratio w is customized by SaaS providers. The expected investment return ($expInvRet_{ijl}^{new}$) is given by Eq. (13):

$$expInvRet_{ijl}^{new} = \omega \times \frac{Cost_{ijl}^{new}}{T_{ijl}^{new}}; \quad \forall i \in I,\ j \in J,\ l \in N_j \tag{13}$$

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* (Step 20). If the maximum investment return is gained by initiating a new VM (Step 22), 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) (Step 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 resource 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. **continue;**
5. }
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}) $\geqslant$ $expInvRet_{ij}^{new}$)
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}.
}

3.2.2. 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 $O(RJ + R^2)$, 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***$(ret_{ij}^{new}) \geqslant expInvRet_{ij}^{new}$)
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}.
}

3.2.3. 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 $O(RJ + R^2)$, 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***$(ret_{ij}^{new}) \geqslant expInvRet_{ij}^{new}$)
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}.
}

4. 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.
- 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. 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 [19] 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. From users' per-

Table 1
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

spective, 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 users. All the parameters from both users' and IaaS providers' side used in the simulation study are given in following subsections:

4.1.1. Users' side

We examine our algorithms with 5000 users. From the user side, five parameters (deadline, service time, budget, arrival rate and penalty rate factor) are varied to evaluate their impact on the performance of our proposed algorithms. Request arrival rate follows poisson distribution as many previous works [37,38] model arrival rate as poisson distribution. Similar as other works, we use a normal distribution to model all parameters (standard deviation $= (1/2) \times$ mean), because there is no available workload specifying these parameters. Eq. (14) is used to calculate the **deadline** (DL_{ijl}^{new}). α is the factor which is used to vary the deadline from "very tight" ($\alpha = 0.5$) to "very relax" ($\alpha = 2.5$). $estprocT_{ijl}^{new}$ indicates the new service request's estimated processing time.

$$DL_{ijl}^{new} = \alpha \times estprocT_{ijl}^{new} + estprocT_{ijl}^{new}; \quad \forall i \in I,\ j \in J,\ l \in N_j \tag{14}$$

- **Service time** is estimated based on the Request Length (*MI*) and the Millions of Instruction per Second (*PS*) of a VM. The mean Request Lengths are selected between 10^6 MI ("very small") to 5×10^6 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 Eq. (1)) is modelled by Eq. (15). 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).

$$\beta = \frac{B^{new}}{DL^{new} \times r}; \quad \forall i \in I,\ j \in J \tag{15}$$

4.1.2. Resource providers' side

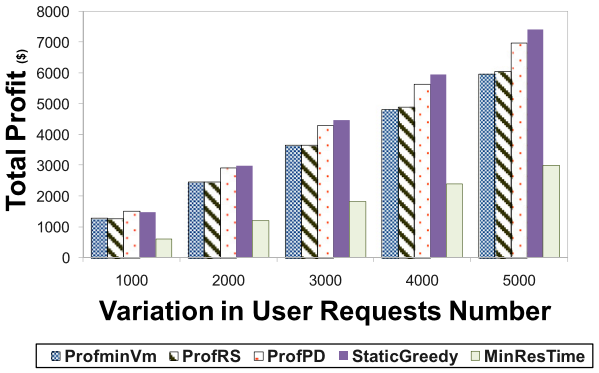
We consider five resource providers – IaaS providers, which are Amazon EC2 [29], GoGrid [31], Microsoft Azure [33], RackSpace [32] and IBM [34]. To simulate the effect of using different VM types, MIPS ratings are used. Thus, a MIPS value of an equivalent processor is assigned to the request processing capability of each VM type. The price schema of VMs follows the price schema of GoGrid [31], Amazon EC2 [29], RackSpace [32], Microsoft Azure [33], and IBM [34]. 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 $= (1/2) \times$ mean). The mean of initiation time is calculated by conducting real experiments of 60 samples on GoGrid [31] and Amazon EC2 [29] done for four days (2 week days and 2 weekend days).

4.2. Performance results

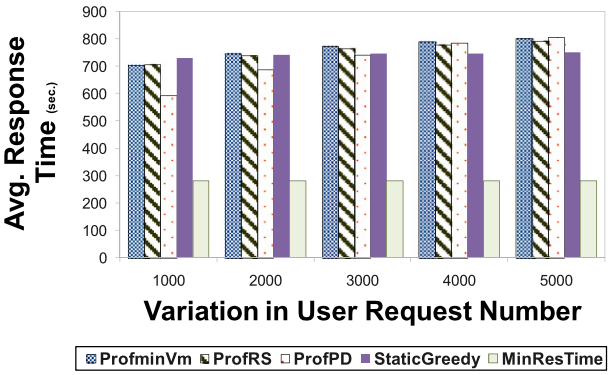
In this section, we first compare our proposed algorithms with reference algorithms by varying number of users. Then, the impact of QoS parameters on the performance metrics is evaluated. Finally, robustness analysis of our algorithm is presented. All of the results present the average obtained by 5 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 $= 4 \times 10^6$ MI, and penalty rate factor (r) = 10.

4.2.1. Comparison with reference algorithms

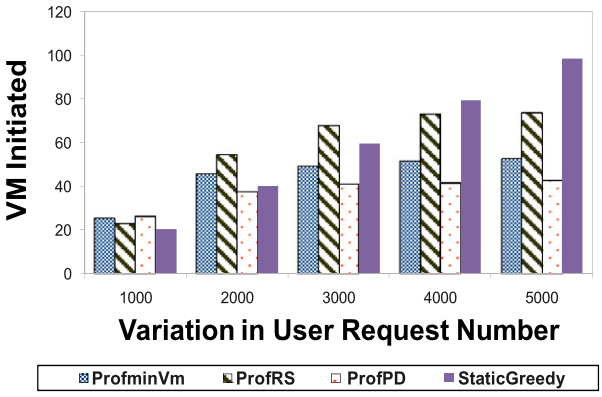
To observe the overall performance of our algorithms, we vary the number of users from 1000 to 5000 without varying other factors such as deadline and budget. Fig. 6 presents the comparison of our proposed algorithms with reference algorithms *StaticGreedy* and *MinResTime* in terms of the four performance metrics. When the number of user requests varies



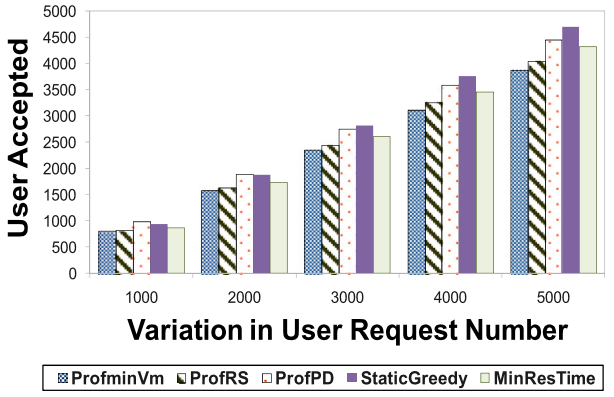
(a). Total profit



(b). Average response time



(c). Number of initiated VMs



(d). Number of accepted users

Fig. 6. Overall algorithms' performance during variation in number of user requests.

from 1000 to 5000, for each algorithm the total profit and average response time has increased, because of more user requests.

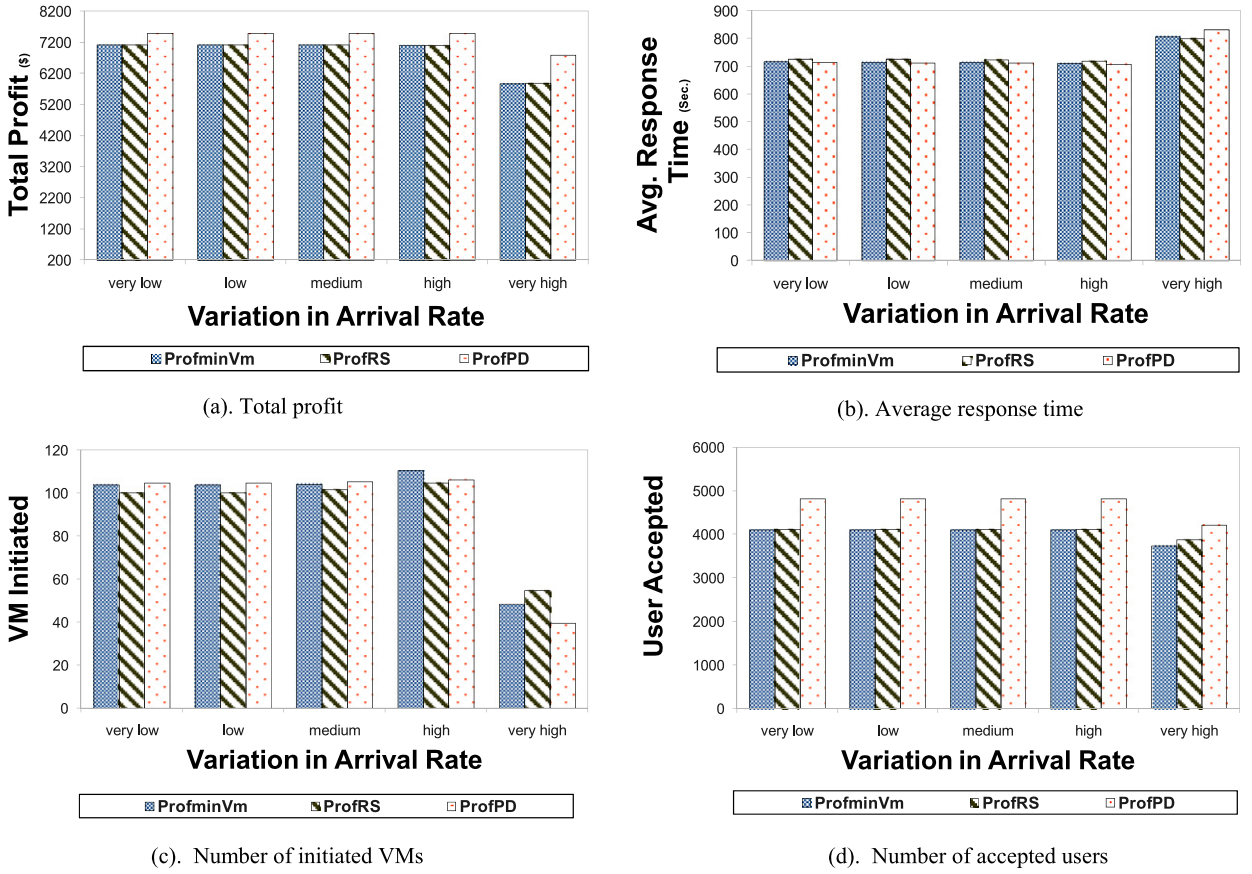
Fig. 6 shows that *ProfPD* earns 8% less profit (Requests = 5000) for SaaS provider than *StaticGreedy* which is used as the upper bound. That is because in the case of *StaticGreedy*, all the user requests are already known from the beginning to the SaaS provider. The base algorithm *MinResTime* has smaller (two third of *StaticGreedy*) response time, but earns less profit (approximately half of *ProfPD*). These observations indicate the trade-off between response time and profit, which SaaS provider has to manage while scheduling requests.

Fig. 6a shows that the *ProfPD* achieves (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 number of users is 1000 *ProfPD* earns 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 became larger. This is because when the number of requests increased, the number of users being accepted increased by utilizing initiated VMs. If all requests are known before scheduling, then *StaticGreedy* is the best choice for maximizing profit, however, in the 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, the *ProfPD* is effective in maximizing profit in heavy workload situations.

Fig. 6b shows that our algorithms' trends of response time increase from 1000 users to 5000 users because of increasing in processing of user requests per VM. When there is smaller number of requests, the difference between different algorithm's response times becomes significant. For example, with 1000 requests, *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 smaller initiation time. *ProfminVM* provides the lowest response time compared to others, because it can serve a new user with new VMs.

4.2.2. 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.



(a). Total profit

(b). Average response time

(c). Number of initiated VMs

(d). Number of accepted users

Fig. 7. Impact of arrival rate variation.

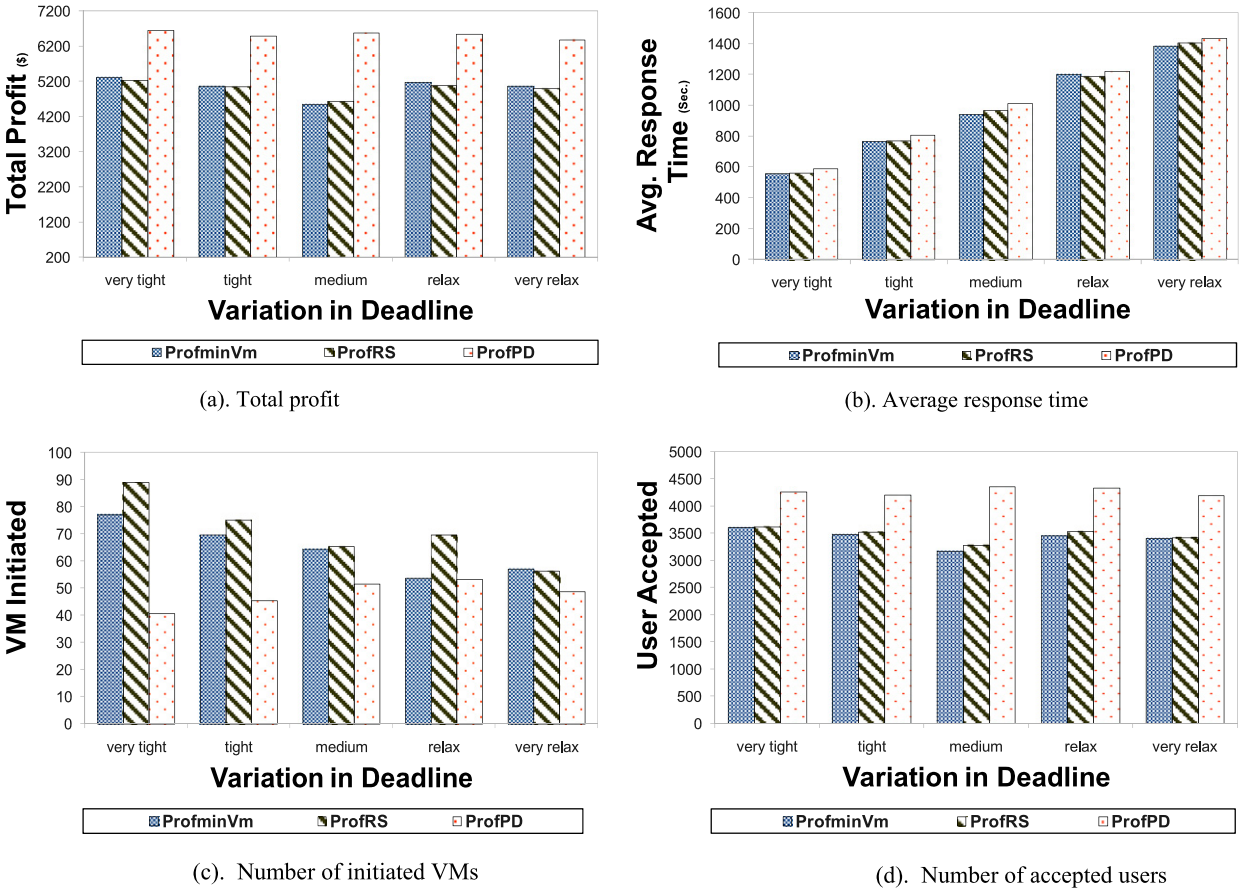
*4.2.2.1. Impact of variation in **arrival rate*** To observe the impact of arrival rate in our algorithms, we vary the arrival rate factor, while keeping all other factors such as deadline, budget as the same. All experiments are conducted with 5000 user requests. It can be seen from Fig. 7 that when 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 user arrival per second, the service capability is decreased due to fewer new VM instantiations.

Fig. 7a 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 increases 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* achieve similar profit, but *ProfRS* accepts 4% more requests with 13% more VMs than *ProfminVM*. Therefore, in this 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 compared to *ProfminVM* because although it provides similar profit as *ProfminVM*, it accepts more requests, leading to market share expanding.

Fig. 7b shows that the *ProfPD* achieves in the smallest 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 times. However, there is a drastic increase in response time when the arrival rate is very high because more requests are 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*.

*4.2.2.2. Impact of variation in **deadline*** 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. Fig. 8a shows that the *ProfPD* achieved the highest profit (45% over *ProfminVM* and 41% over *ProfRS*) by accepting 33% more user requests (Fig. 8d) and initiating 52% less VMs (Fig. 8c). 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 in this scenario.

Fig. 8b shows that when deadline is relaxed, *ProfPD* results in 4% higher average response time than in the case of *ProfminVM* and *ProfRS*. The *ProfPD* has larger response time because of the two factors governing response time, i.e., request's



(a). Total profit

(b). Average response time

(c). Number of initiated VMs

(d). Number of accepted users

Fig. 8. Impact of deadline variation.

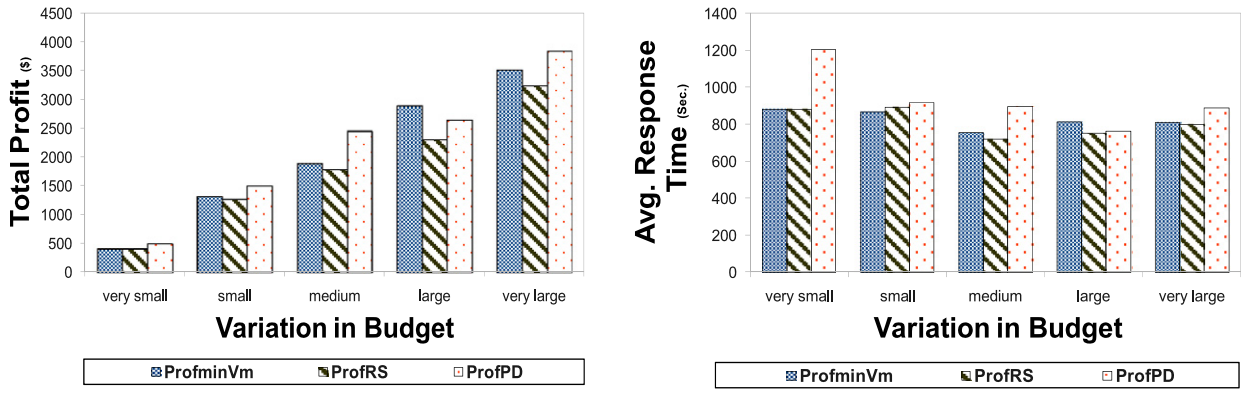
service time and VM initiation time. It can be seen from Fig. 8d that *ProfPD* always requires less VMs, to process more requests. Thus, when service time is comparable to the VM initiation time, the response time will be lower. When the VM initiation time is larger than the service time, the response time is affected by the number of initiated VMs.

4.2.2.3. Impact of variation in ***budget*** Fig. 9 shows variation of budget impacts our algorithms, while keeping all other factors such as arrival rate and deadline fixed. Fig. 9a shows that when budget is varies from "very small" to "very large", in average the total profit by all the algorithms has increased, and response time has decreased since less requests are processed using more VMs. From Fig. 9a, 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", *ProfminVM* provides the highest profit (20%) over other algorithms by accepting similar number of requests while initiating more VMs without penalty delay. This is due to an increase in the *Penalty Delay Rate* (β) (Eq. (15)) with the budget raise. Between *ProfminVM* and *ProfRS*, *ProfminVM* provides more profit in all scenarios. Therefore, in this scenario a SaaS provider should consider *ProfPD*, *ProfminVM* compared with *ProfRS*.

In the case of response time (Fig. 9b), *ProfPD* on average delayed the processing of request for the longest time (e.g. 33% bigger 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 caused by the contribution of VM initiation time which becomes very significant when *ProfRS* initiated large number of VMs.

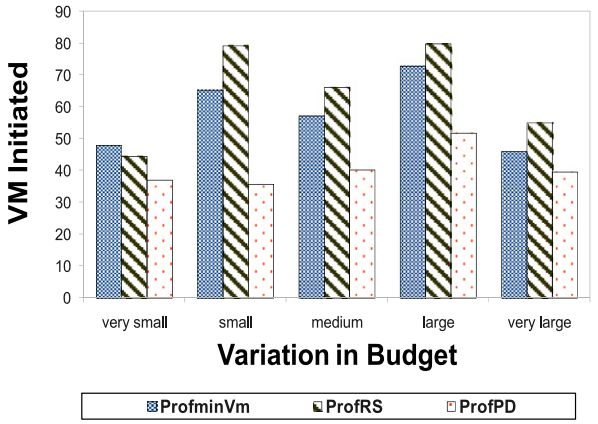
4.2.2.4. Impact of variation in ***service time*** Fig. 10 shows how service time impacts our algorithms, while keeping all other factors such as arrival rate and deadline as the same. In order to vary the service time, five classes of request length (*MI*) are chosen from "very small" (10^6 MI) to "very large" (5×10^6 MI).

Fig. 10a shows that the total profit by all algorithms has slightly decreased but response time increased rapidly when the request length varies from "very small" to "very large". *ProfPD* achieves the highest profit among other 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 requests (Fig. 10d) and initiating 32% (Fig. 10c) less VMs. In addition, *ProfminVM* and *ProfRS* achieve similar profit in most of the cases. Therefore, the *ProfPD* is the best solution for any size of requests.

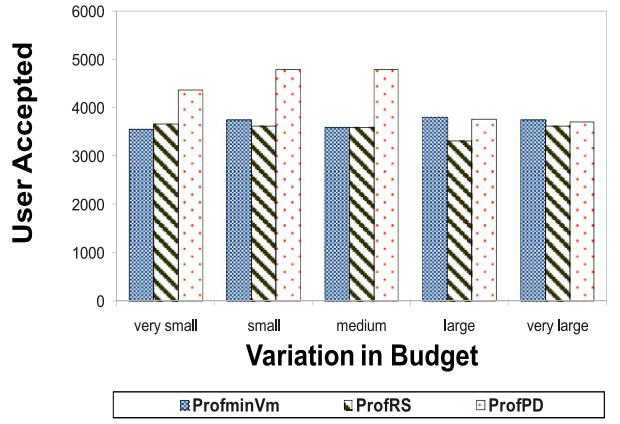


(a). Total profit

(b). Average response time



(c). Number of initiated VMs



(d). Number of accepted users

Fig. 9. Impact of budget variation.

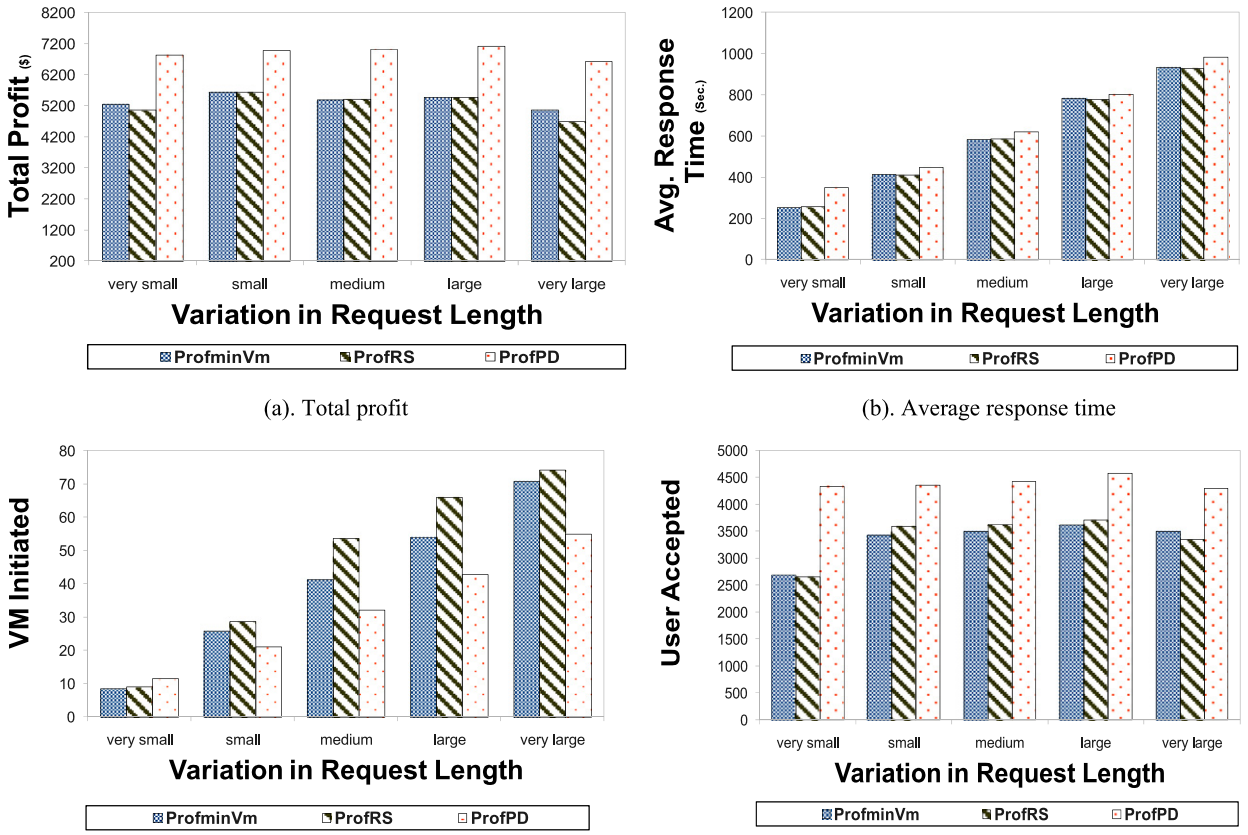
In addition, it can be observed from Fig. 10b that *ProfPD* provides only a slightly higher response time (almost 6%) than others except when the request size is very small. When request size is very small, the response time provided by *ProfPD* becomes 27% bigger than others, because it accepts 63% more user requests with 22% more VMs, leading to more requests waiting for processing on each VM.

4.2.2.5. Impact of variation in ***penalty rate*** In this section, we investigate how penalty rate (β) impacts our algorithms. The penalty rate (Eq. (15)) 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 Fig. 11.

In can be observed from Fig. 11 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. The total profit (Fig. 11a) and average response time (Fig. 11b) 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 requests and initiating 30% less VMs. In addition, when the penalty rate varies from "very low" to very high", the response time slightly decreased. This is because *ProfPD* accepts a little bit less requests with similar number of VMs. Thus, the number of requests waiting in each VM becomes smaller, leading to faster response time for each request.

4.2.2.6. Impact of variation in ***initiation time*** In this section, we analyse the variation of initiation time impacts our algorithms. Fig. 12a illustrates that with increase in initiation time the total profit achieved by all the algorithms decreases slightly while response time has increased a little bit. Due to increase in initiation time, the number of initiated VMs (Fig. 12c) has decreased rapidly due to the contribution of initiation time in SaaS providers cost (spending). In all the scenarios, *ProfPD* achieves highest profit over others by accepting 17% more requests (Fig. 12d) and with 37% less initiated VMs. Therefore, *ProfPD* is the best choice for a SaaS provider in this scenario.

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 response time of *ProfPD* is the lowest in this scenario; because of large initiation time of VM, the response time is



(a). Total profit

(b). Average response time

(c). Number of initiated VMs

(d). Number of accepted users

Fig. 10. Impact of request length variation.

also increased with each initiated VM. However, the contribution to delay in processing of requests, due to more number of requests per VM also increases. This leads to higher response time in the scenario when the initiation time is "very long".

4.3. Robustness analysis

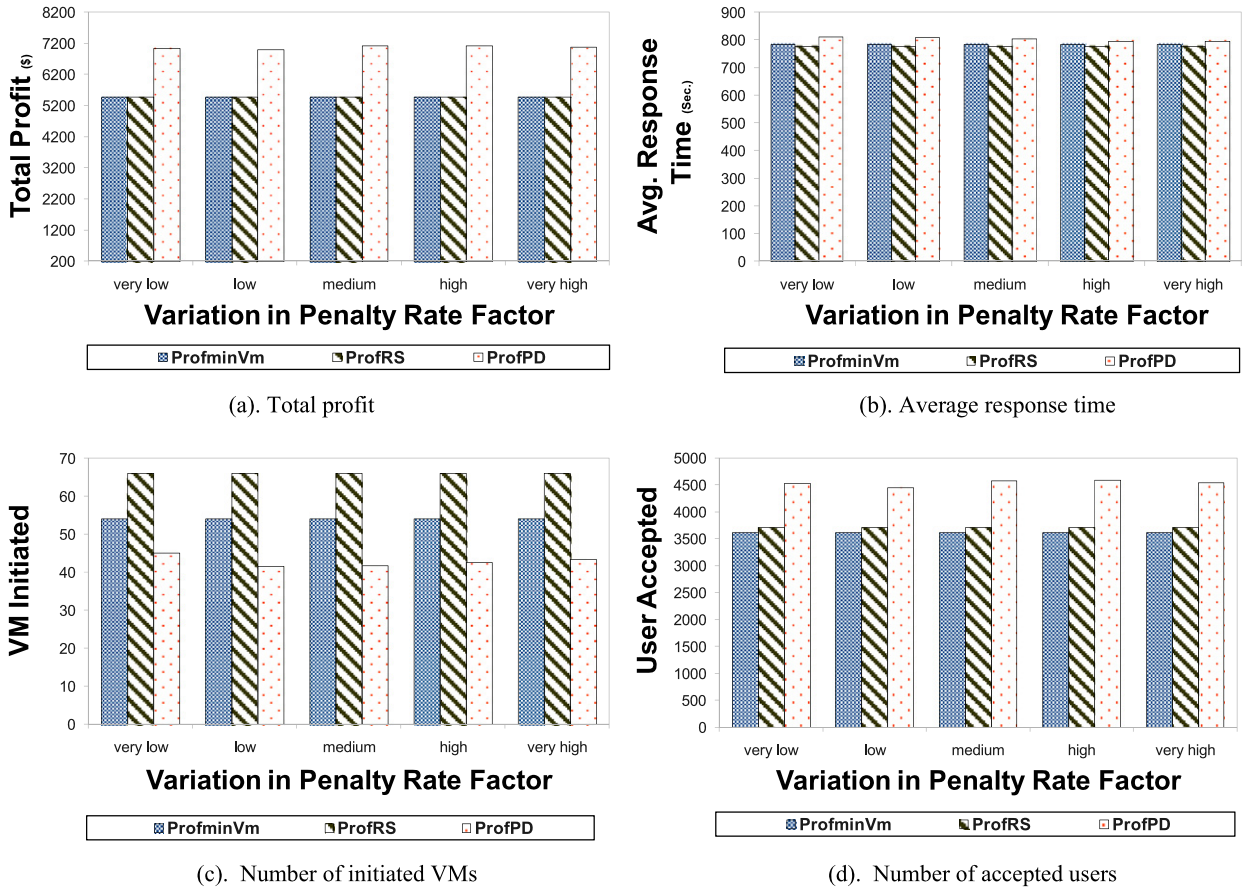
In order to evaluate the robustness of our algorithms, we run some experiments by reducing the actual performance of VMs in the SLA(R) promised by IaaS providers. This performance degradation has been observed by previous research study in Cloud computing environments [35]. This experiment is conducted also to justify the inclusion of compensation (penalty) clauses in SLAs which is absent in current IaaS providers' SLAs [30]. We modelled the reduced performance using a normal distribution with average variation between mean varies 0% and 50%.

Fig. 13 shows that during the degradation of VM performance, the average total profit (Fig. 13a) has reduced 11% and average response time (Fig. 13b) 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.

Two solutions to handle this VMs performance degradation are: first, utilization of the penalty clause in SLA(R) to compensate for profit loss; second, considering the degradation as a potential risk. Therefore, during the scheduling process a (300 seconds) slack time is added in estimated service processing time and it can be seen from Fig. 14, that the latter solution reduces 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.

5. 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 [15] or designed for fixed number of resources, such as FirstPrice [4] and FirstProfit [7]. In Cloud, IaaS providers focusing on maximize profit and many works [26,15,3] proposed market based scheduling approaches. For instance, Amazon [29] introduced spot instance way for customers to



(a). Total profit

(b). Average response time

(c). Number of initiated VMs

(d). Number of accepted users

Fig. 11. Impact of penalty rate factor variation.

buy those unused resources at bargain prices. This is a way of optimizing resource allocation if customers are happy to be terminated at any time. However, our goal is not only to maximize profit but also satisfy the SLA agreed with the customer.

At platform category, Projects such as InterCloud [16], Sky Computing [18], and Reservoir [17] investigated the technological advancement that is required to aid the deployment of cloud services across multiple infrastructure providers. However, research at the SaaS provider level is still in its infancy, because many works do not consider maximizing profit and guaranteeing SLA with the leasing scenario from multiple IaaS providers, where resources can be dynamically expanded and contracted on demand.

In this section, since we focus on developing admission control and scheduling algorithms and strategies for SaaS providers in Cloud, we divide related work into two sub-sections: admission control and scheduling.

5.1. Admission control

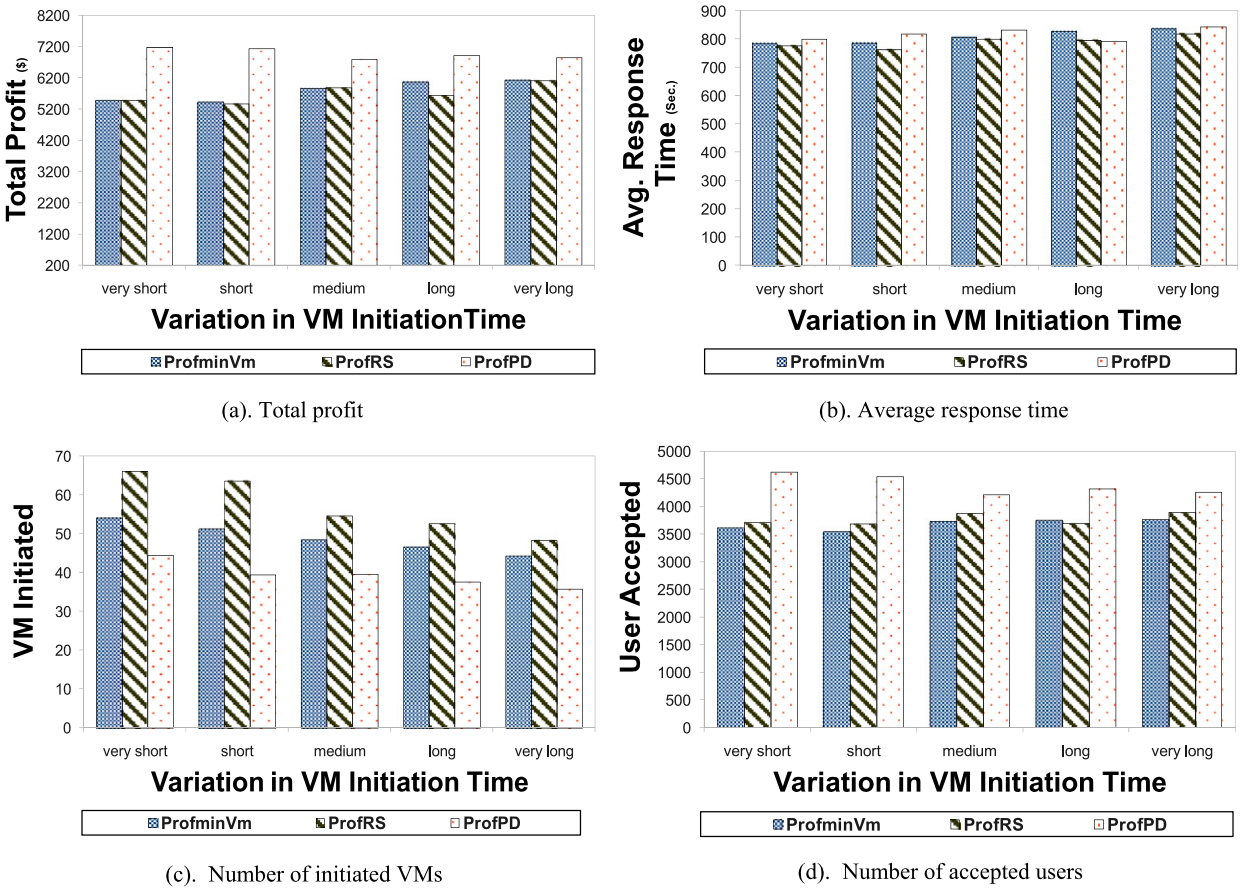
Yeo and Buyya presented algorithms to handle penalties in order to enhance the utility of the cluster based on SLA [1]. Although they have outlined a basic SLA with four parameters in cluster environment, multiple resources and multiple QoS parameters from both user and provider sides are not explored.

Bichler and Setzer proposed an admission control strategy for media on demand services, where the duration of service is fixed [12]. Our approach allows 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 increase the profit in 3 times will be accepted.

Islam et al. investigated policies for admission control that consider jobs with deadline constraints and response time guarantees [27,28]. 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.

Jaideep and Varma proposed learning-based admission control in Cloud computing environments [2]. Their work focuses on the accuracy of admission control but does not consider software service providers' profit.

Reig et al. contributed on minimizing the resource consumption by requests and executing them before their deadline with a prediction system [23]. Both the works use deadline constraint to reject some requests for more efficient scheduling. However, we also consider the profit constraint to avoid wastage of resources on low profit requests.



(a). Total profit

(b). Average response time

(c). Number of initiated VMs

(d). Number of accepted users

Fig. 12. Impact of initiation time variation.

5.2. Scheduling

Chun et al. built a prototype cluster of time-sharing CPU usage to serve user requests [13]. A market-based approach to solve traffic spikes for hosting Internet applications on Cluster was studied by Coleman et al. [14,13]. Lee et al. investigated a profit-driven service request scheduling for workflows [3]. These related works focus on scenarios with fixed resources, while we focus on scenarios with variable resources.

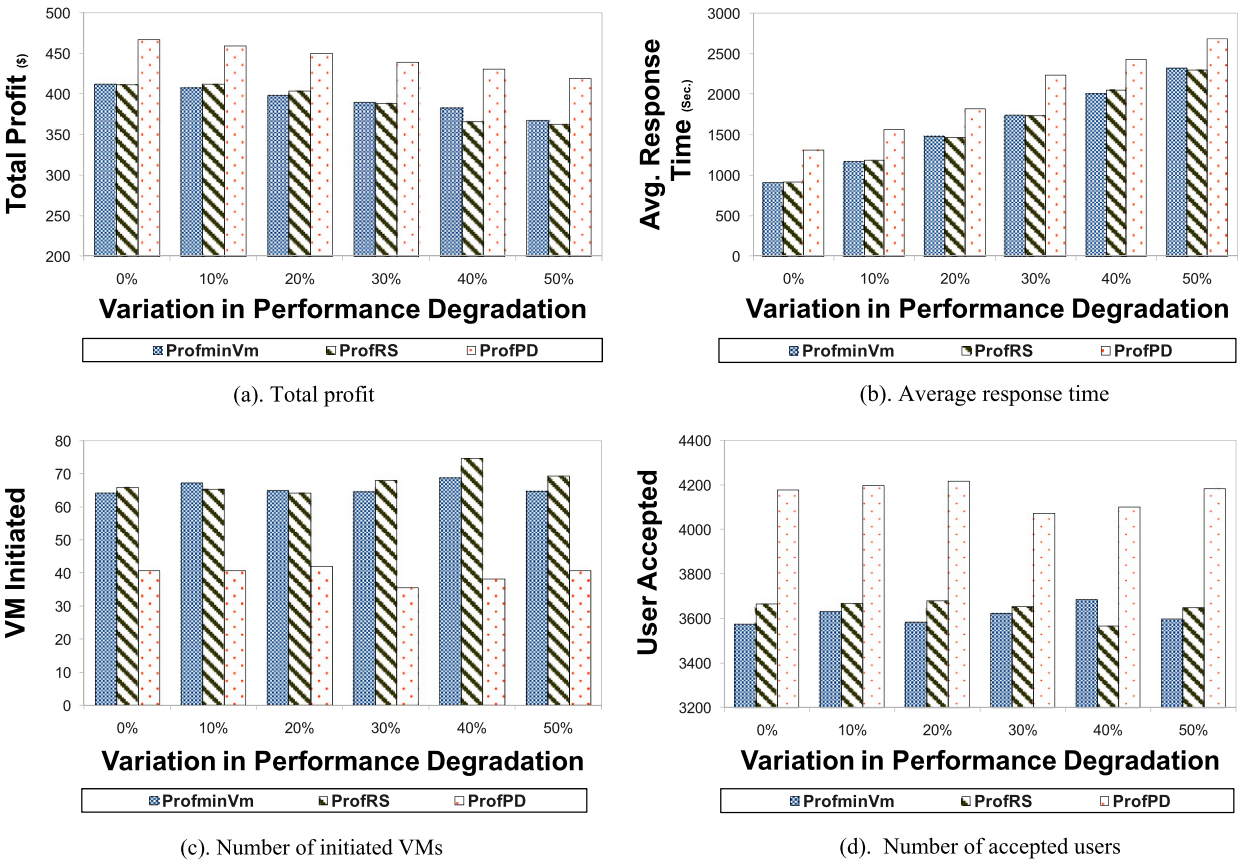
Liu et al. analysed the problem of maximizing profit in e-commerce environment using web service technologies, where the basic distributed system is Cluster [21]. Kumar et al. investigated two heuristics, HRED and HRED-T, to minimize business value but they studied only the minimization of cost [36]. Garg et al. also proposed time and cost based resource allocation in Grids on multiple resources for parallel applications [26]. However, our current study uses different QoS parameters, (e.g. penalty rate). In addition, our current study focuses on Clouds, where the unit of resource is mostly VM, which may consist of multiple processors.

Menasce et al. proposed a priority schema for requests scheduling based on user status. The algorithm assigns higher priority to requests with shopping status during scheduling to improve the revenue [22]. Nevertheless, their work is not SLA-based and response time is the only concern.

Xiong et al. 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 [24]. We consider different QoS parameters (i.e., budget, deadline, and penalty rate), admission control and resource allocation, and multiple IaaS providers. Netto et al. considered deadline as their only QoS parameter for bag-of-task applications in utility computing systems considering multiple providers [25]. Popovici et al. mainly focused on QoS parameters on resource provider's side such as price and offered load [7]. However, our work differs on QoS parameters from both users' and SaaS providers' point of view, such as budget, deadline, and penalty rate.

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.**



(a). Total profit

(b). Average response time

(c). Number of initiated VMs

(d). Number of accepted users

Fig. 13. Impact of performance degradation variation.

- **Our strategies adapt to dynamic resource pools and consistently evaluate the profit of adding a new instance or removing instances, while most previous work deal with fixed size resource pools.**

6. Conclusions and future directions

We presented admission control and scheduling algorithms for efficient resource allocation to maximize profit and customer 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. 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 2.

In this work, we have assumed that the estimated service time is accurate since existing performance estimation techniques (e.g. analytical modelling [20], empirical, and historical data [21]) can be used to predict service times on various types of VMs. However, still some error can exist in this estimated service time [35] 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. 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.

Acknowledgments

This work is supported by the Australian Research Council (ARC) via Discovery/Linkage Project grants. We thank our colleagues especially Rodrigo N. Calheiros and reviewers for their comments on improving the paper.

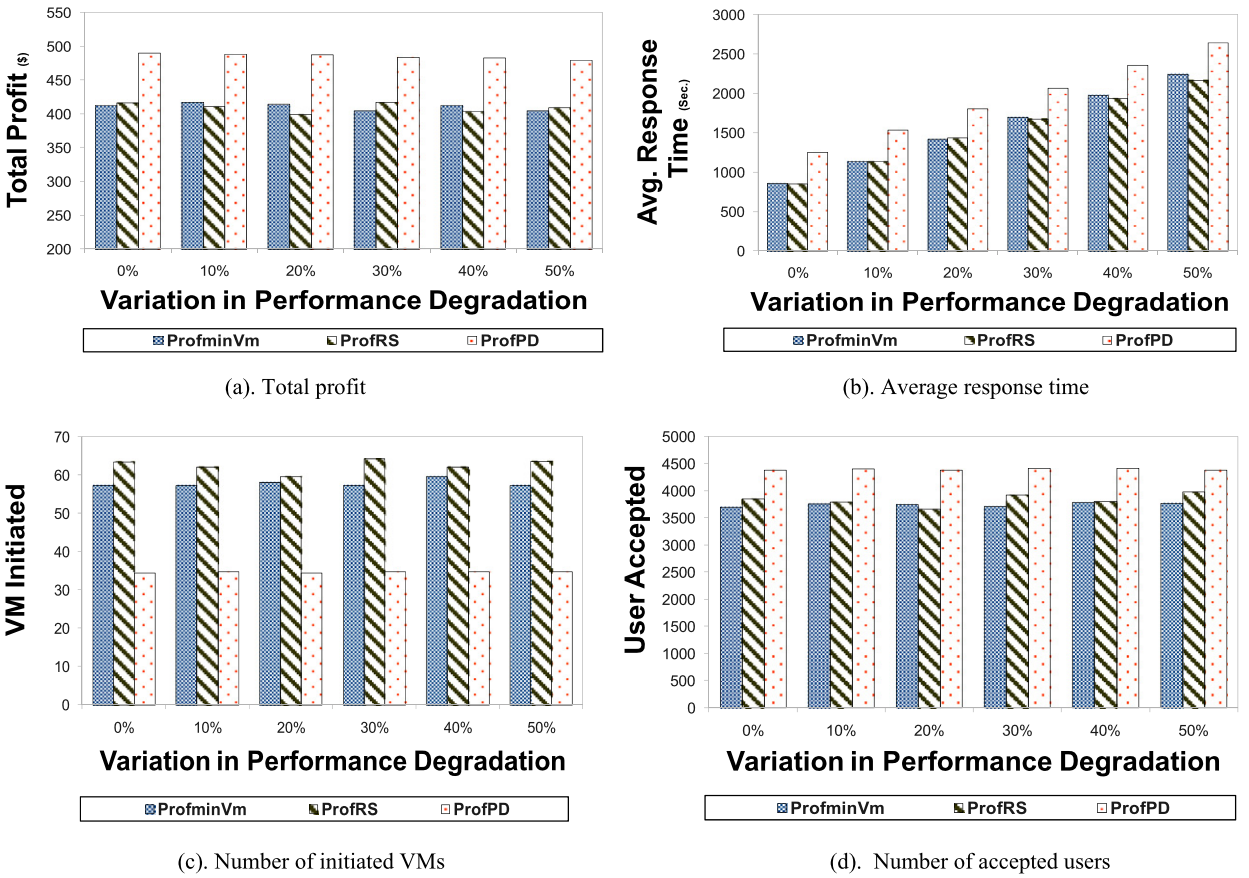


Fig. 14. Impact of performance degradation variation after considering slack time.

Table 2
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 high)	Okay (very high)	Okay (very low)	Okay	No effect	Good (low-high)	Okay
ProfPD	$O(KJ+K^2)$	Best	Best	Best	Best	Best	Best	Best

References

[1] C.S. Yeo, R. Buyya, Service level agreement based allocation of cluster resources: Handling penalty to enhance utility, in: Proceedings of the 7th IEEE International Conference on Cluster Computing (Cluster 2005), Boston, MA, USA, 2005.
[2] 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.
[3] Y.C. Lee, C. Wang, A.Y. Zomaya, B.B. Zhou, Profit-driven service request scheduling in clouds, in: Proceedings of the International Symposium on Cluster and Grid Computing (CCGrid 2010), Melbourne, Australia, 2010.
[4] O.F. Rana, M. Warnier, T.B. Quillinan, F. Brazier, D. Cojocarasu, Managing violations in service level agreements, in: Proceedings of the 5th International Workshop on Grid Economics and Business Models (GenCon 2008), Gran Canaria, Spain, 2008.
[5] D.E. Irwin, L.E. Grit, J.S. Chase, Balancing risk and reward in a market-based task service, in: Proceedings of the 13th International Symposium on High Performance Distributed Computing (HPDC 2004), Honolulu, HI, USA, 2004.
[6] 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.
[7] I. Popovici, J. Wiles, Profitable services in an uncertain world, in: Proceedings of the 18th Conference on Supercomputing (SC 2005), Seattle, WA, 2005.
[8] R. Buyya, C.S. Yeo, S. Venugopal, J. Broberg, I. Brandic, Cloud computing emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility, Future Gener. Comput. Syst. 25 (6) (2009) 599–616, Elsevier Science, Amsterdam, The Netherlands.

[9] L.M. Vaquero, L. Rodero-Merino, J. Caceres, M. Lindner, A break in the clouds: Towards a cloud definition, ACM SIGCOMM Comput. Commun. Rev. 39 (1) (2009) 50–55.
[10] D. Parkhill, The Challenge of the Computer Utility, Addison–Wesley, USA, 1966.
[11] M.A. Vouk, Cloud computing-issues, research and implementation, in: Proceedings of 30th International Conference on Information Technology Interfaces (ITI 2008), Dubrovnik, Croatia, 2008.
[12] M. Bichler, T. Setzer, 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, 2007.
[13] N.B. Chun, D.E. Culler, 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, 2002.
[14] 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.
[15] J. Broberg, S. Venugopal, R. Buyya, Market-oriented grids and utility computing: The state-of-the-art and future directions, Journal of Grid Computing 3 (6) (2008) 255–276.
[16] 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.
[17] B. Rochwerger, et al., The reservoir model and architecture for open federated cloud computing, IBM Syst. J. 4 (53) (2009) 1–11.
[18] K. Keahey, A. Matsunaga, J. Fortes, Sky computing, IEEE Internet Computing 13 (5) (2009) 43–51.
[19] R.N. Calheiros, R. Ranjan, A. Beloglazov, C.A.F. De Rose, R. Buyya, CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms, Software: Practice and Experience (ISSN 0038-0644) 1 (41) (2011) 23–50, Wiley Press, New York, USA.
[20] Animoto, retrieved on 12 Sep. 2010, http://developer.amazonwebservices.com/connect/entry!default.jspa;jsessionid=5E63CF198EA2949E920D500303C858CE?categoryID=89&externalID=932&fromSearchPage=true.
[21] Z. Liu, M.S. Squillante, J.L. Wolf, On maximizing service-level-agreement profits, in: Proceedings of the 3rd ACM Conference on Electronic Commerce (EC 01), Tampa, Florida, USA, 2001.
[22] 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.
[23] G. Reig, J. Alonso, J. Guitart, Deadline constrained prediction of job resource requirements to manage high-level SLAs for SaaS cloud providers, Tech. Rep. UPC-DAC-RR, Dept. d'Arquitectura de Computadors, University Politécnica de Catalunya, Barcelona, Spain, 2010.
[24] K. Xiong, H. Perros, SLA-based resource allocation in cluster computing systems, in: Proceedings of 17th IEEE International Symposium on Parallel and Distributed Processing (IPDPS 2008), Alaska, USA, 2008.
[25] M. Netto, R. Buyya, Offer-based scheduling of deadline-constrained bag-of-tasks applications for utility computing systems, in: 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, 2009.
[26] 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.
[27] M. Islam, P. Balaji, P. Sadayappan, D.K. Panda, 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.
[28] 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.
[29] J. Varia, Architecting applications for the Amazon Cloud, in: R. Buyya, J. Broberg, A. Goscinski (Eds.), Cloud Computing: Principles and Paradigms, Wiley Press, New York, USA, ISBN: 978-0470887998, 2010, http://aws.amazon.com.
[30] CIO, retrieved on 10 Sep. 2010, http://www.cio.com.au.
[31] GoGorid, retrieved on 10 Sep. 2010, http://www.gogrid.com.
[32] RackSpace, retrieved on 10 Sep. 2010, http://www.rackspacecloud.com.
[33] Microsoft Azure retrieved on 10 Sep. 2010, http://www.microsoft.com/windowsazure/.
[34] IBM, retrieved on 10 Sep. 2010, http://www.ibm.com/ibm/cloud/ibm_cloud/.
[35] S. Ostermann, A. Iosup, M.N. Yigitbasi, R. Prodan, T. Fahringer, D. Epema, 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, 2009.
[36] 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).
[37] M.L. McManus, M.C. Long, A. Copper, E. Litavak, Queuing theory accurately models the need for critical care resources, Anesthesiology (ISSN 0003-3022) 100 (5) (2004) 1271–1276, Lippincott Williams & Wilkins, USA.
[38] R.W. Wolff, Poisson arrivals see time averages, Oper. Res. 30 (2) (1998) 223–231.
[39] Saleforce.com, retrieved on 10 Sep. 2010, http://www.salesforce.com/au/.
[40] S. Ostermann, A. Iosup, N. Yigitbasi, R. Prodan, T. Fahringer, D. Epema, A performance analysis of EC2 cloud computing services for scientific computing, in: Proceedings of 1st International Conference on Cloud Computing (CloudComp), Munich, Germany, 2009.