SLA-Based Resource Provisioning for Hosted Software-as-a-Service Applications in Cloud Computing Environments

Linlin Wu, Saurabh Kumar Garg, Steve Versteeg, and Rajkumar Buyya

Abstract—Cloud computing is a solution for addressing challenges such as licensing, distribution, configuration, and operation of enterprise applications associated with the traditional IT infrastructure, software sales and deployment models. Migrating from a traditional model to the Cloud model reduces the maintenance complexity and cost for enterprise customers, and provides on-going revenue for Software as a Service (SaaS) providers. Clients and SaaS providers need to establish a Service Level Agreement (SLA) to define the Quality of Service (QoS). The main objectives of SaaS providers are to minimize cost and to improve Customer Satisfaction Level (CSL). In this paper, we propose customer driven SLA-based resource provisioning algorithms to minimize cost by minimizing resource and penalty cost and improve CSL by minimizing SLA violations. The proposed provisioning algorithms consider customer profiles and providers' quality parameters (e.g., response time) to handle dynamic customer requests and infrastructure level heterogeneity for enterprise systems. We also take into account customer-side parameters (such as the proportion of upgrade requests), and infrastructure-level parameters (such as the service initiation time) to compare algorithms. Simulation results show that our algorithms reduce the total cost up to 54 percent and the number of SLA violations up to 45 percent, compared with the previously proposed best algorithm.

Index Terms—Cloud computing, Service Level Agreement (SLA), resource allocation, scheduling, software as a service, customer-driven, Key Performance Indicator (KPI), resource provisioning

1 Introduction

Cloud computing has emerged as a new paradigm for delivery of applications, platforms, and computing resources (processing power/bandwidth/storage) to customers in a ''pay-as-you-go-model''. Cloud computing falls into three categories: Software as a Service (SaaS), Platform as a Service (PaaS) and Infrastructure as a Service (IaaS). SaaS Clouds provide software services to end users. IaaS Clouds provide a virtual computing environment, where computing capacity is delivered by assigning Virtual Machines (VMs) to IaaS users on demand. In the middle of SaaS and IaaS, the PaaS Clouds provide application development, deployment tools and execution management services. Prior to the Cloud and in the early days of Web-based enterprise application deployment, the administration task was easy since the single important objective of resource provisioning was the performance, such as the time spent on resource provisioning [23]. Over the course of time, the complexity of applications has grown, which has increased the difficulties in their administration. Accordingly, enterprises have realized that it is more efficient to outsource some of their applications to third-party SaaS providers enabled by Cloud computing due to the following reasons [18]:

- It reduces the maintenance cost, because along with the growth in the complexity, the level of sophistication required to maintain the system has increased dramatically.
- By using SaaS, enterprises do not need to invest in expensive software licenses and hardware upfront without knowing the business value of the solution.

Therefore, by moving to the SaaS model customers benefit from continuously maintained software. The complexity of transitioning to new releases is managed transparently by the SaaS providers. Thanks to the flexibility, scalability and cost-effectiveness of the SaaS model, it has been increasingly adopted for distributing many enterprise software systems, such as banking and e-commerce business software [7], [9]. In this scenario, enterprises need to establish a Service Level Agreement (SLA) with SaaS providers. SLA is a legal contract between participants to ensure that their Quality of Service (QoS) requirements are met and if any party violates the SLA terms, the defaulter has to pay penalty according to the clauses defined in the SLA.

To guarantee SLAs, enterprise software providers (such as Compiere ERP) in the industry allocate dedicated VMs for each customer [17], so they can ensure software response time. However, this can cause wastage of hardware resources due to the under-utilized resources at non-peak load as discussed in our previous work [22].

- *L. Wu and R. Buyya are with the Cloud Computing and Distributed Systems (CLOUDS) Laboratory, Department of Computing and Information Systems, The University of Melbourne, Australia.*
- *S.K. Garg is with the Department of Computing and Information System, Faculty of Engineering and ICT, University of Tasmania, Hobart, TAS, 7001, Australia. E-mail: Saurabh.Garg@utas.edu.au.*
- *S. Versteeg is with CA Technologies, Melbourne, Australia.*

Manuscript received 26 Oct. 2011; revised 28 June 2012; accepted 10 Oct. 2013. Date of publication 3 Nov. 2013; date of current version 17 Sept. 2014. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TSC.2013.49

TABLE 1
The Summary of Penalty Delay Time According to Request Types

Response Time	First Time Rent-ftr	Upgrade Service	
		Add account-addAcc	Upgrade product-upServ
Defined in SLA	respT (ftr)	respT(upServ, addAcc)	respT (upServ, upPro)
Actual Time	iniT	iniT + totalDTT	iniT +totalDTT

Research publications related to SLA-based cost optimisation and Customer Satisfaction Level (CSL) maximization for SaaS providers are still in their preliminary stages, and current publications in Cloud computing [2], [10], [13] are focused mostly on market oriented models for IaaS providers. Many authors do not consider customer driven management, where resources have to be dynamically reallocated according to the customer's on-demand requirements. Our previous work [22] proposed algorithms to minimize the total cost and SLA violations without considering a customer's profile (such as company size), which is important for SaaS providers to reserve resources based on this information.

Our focus, with respect to CSL, is on delivering acceptable response times. CSL is impacted by SLA violations, when the pre-defined response time in the SLA is violated by the provider. The SLA violation causes penalty. The service quality improvement (SQI) is defined as how much faster the actual response time compares to the minimum response time documented in the SLA. To maximize the CSL, we design algorithms to minimize SLA violations by resource reservation and requests rescheduling. In addition, the SQI will also impact the CSL but this is a minor factor because many customers may do not consider whether the quality provider offer is better than expected. Therefore we did not consider this in our algorithms but only plotted the figures related to it in experiments.

This paper proposes customer driven heuristic algorithms to minimize the total cost (including infrastructure and penalty cost) by resource provisioning. These algorithms also take into account customer profiles (such as their credit level) and multiple Key Performance Indicator (KPI) criteria. A holistic way to quantify the customer experience is by considering KPIs from seven categories: Financial, Agility, Assurance, Accountability, Security and Privacy, Usability and Performance [23]. To improve a SaaS application's performance quality rating, we consider three KPIs, including one from provider's perspective: cost (part of the Financial category) and two from customers' perspective: service response time (part of the Performance category) and SLA violations (related to Assurance):

- Cost: the total cost including VM and penalty cost.
- Service response time: how long it takes for users to receive a response. Four types of response time are summarized in Table 1.
- SLA violations: the possibility of SLA violations creates a risk for SaaS providers. In this paper, SLA violations are caused by elapse in the expected response time, and whenever a SLA violation occurs, a penalty is charged.

To satisfy customer requests to minimize the total cost and SLA violations for SaaS providers, the following key questions are addressed:

- How to manage dynamic customer demands? (such as upgrading from a standard product edition to an advanced product edition or adding more accounts)
- How to reserve resources by considering the customer profiles and multiple KPI criteria?
- How to map customer requirements to infrastructure level parameters?
- How to deal with the infrastructure level heterogeneity (such as different VM types and service initiation time)?

The **key contributions** of this paper are:

1. Design of a resource provisioning model for SaaS Clouds considering customer profiles and multiple KPI criteria. These considerations are important for resource reservation strategies to improve the CSL.
2. Development of innovative scheduling algorithms to minimize the total cost and number of SLA violations.
3. Extensive evaluation of the proposed algorithms with new QoS parameters such as credit levels.

The rest of the paper is organized as follows. Section 2 presents the detailed scenario from both customers' and providers' perspective, outlines mathematical models, explains mapping strategy, and describes the problem definition. Section 3 describes a reference algorithm ($BestFit$) and two proposed advanced optimization algorithms ($BFResvResource$ and $BFReschedReq$), and a lower bound of the problem. Section 4 presents experimental methodologies including the testbed and evaluation metrics, discusses the overall comparison among performance evaluation results, and compares the algorithms by providing insights into when to apply which algorithm. Section 5 discusses prior research papers related to SLA-based market driven resource allocation in Grid and Cloud computing to identify the novelty of our work. Section 6 concludes the paper by summarizing the comparison results and future directions.

2 System Model

The SaaS model for serving customers in the Cloud is shown in Fig. 1. Customers submit requests for utilizing a Web-based enterprise application service offered by a SaaS provider. The SaaS provider uses a three layered Cloud model, namely the application layer, the platform layer and the infrastructure layer, to satisfy the customer requests.

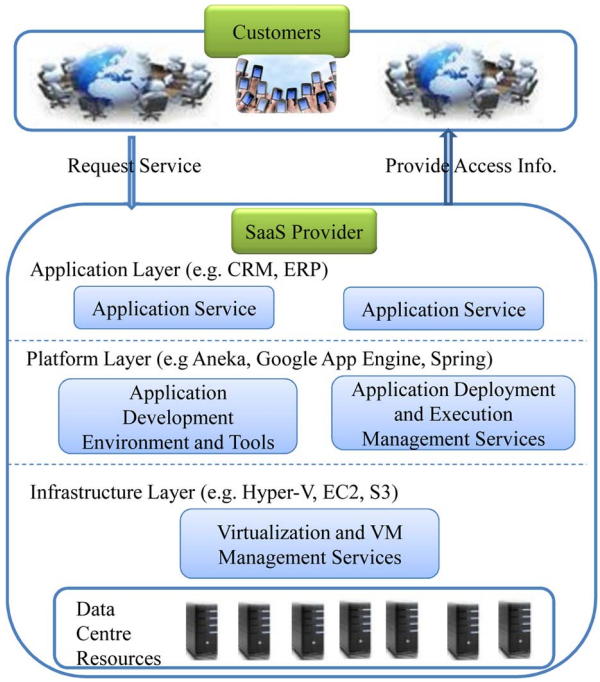

Fig. 1. A system model of SaaS layer structure.

The *application layer* manages all the secured application services, such as the Customer Relationship Management (CRM) or Enterprise Relationship Package (ERP) applications, that are offered to customers by the SaaS provider. The *platform layer* is responsible for application development and deployment (such as Aneka [14], Google App Engine [45], Spring framework). In our model, the function of this layer also includes mapping and scheduling policies for translating the customer side QoS requirements to infrastructure level parameters. The mapping policy considers customer profiles and KPI criteria to measure the SaaS provider's QoS.

The *infrastructure layer* includes the virtualization VM management services (such as VMWare [47], Hyper-V [46]) and controls the actual initiation and termination of VMs resources, which can be leased from IaaS providers, such as Amazon EC2, S3 [14] or own private virtualized clusters. In both cases, the minimization of the number of VMs will deliver savings for the providers.

2.1 Actors

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

2.1.1 SaaS Providers

SaaS providers lease web-based enterprise software as services to customers. The main objective of SaaS providers is to minimize cost and SLA violations. We achieve this objective by proposing customer-driven SLA-based resource provisioning algorithms for Web-based enterprise applications. In our context, a SaaS service provider X offers CRM or ERP software packages with three product editions (for example, Standard, Professional and Enterprise) and each product edition with a fixed price. The current SaaS providers, such as 'Compiere ERP', use a similar service model [15]. In this service model, when a customer *Company* Y submits its 'first time rent' request with a product edition (*Standard*), and additional number of accounts, the SaaS provider needs to allocate resources and then provides the login information to the customer. *Company* Y may require an upgrade in their service by adding additional user accounts or an upgrade of the software edition. In this case, sometimes a new VM is created and the content from the previous VM is migrated to the new one. In practice, the provider has to handle these on-demand customer requests in line with the SLA. The SLA properties including the provider's pre-defined parameters and the customer specified QoS parameters are as follows:

- Product Edition (p): It is defined as the software product package that is offered to customers. For example, *SaaSX* offers Standard, Professional, and Enterprise product editions.
- Request Type (j): This defines the type of customer request, which may be a 'first time rent' or a 'service upgrade' request. 'First time rent' means the customer is renting a new service from this SaaS provider. A 'service upgrade' includes two types of upgrade, which are 'add account' and 'upgrade product'. To downgrade a service, first, the customer needs to terminate the current contract, and then processing of this downgrade request will be treated as a new request.
- Contract Length (cl): How long the customer is going to use the software service.
- Number of Accounts (a): The actual number of user accounts that a customer wants to create. The maximum number of accounts is related to and restricted by the type of product edition.
- Number of Records (n): The average number of records that a customer is able to create for each account during a transaction and this may impact the data transfer time during the service upgrade (The value of this parameter is predefined in the SLA).
- Response Time $(respT)$: It represents the time taken by the provider to process a particular customer request. For example, An SLA violation occurs when the actual response time is longer than it was defined in the SLA. We consider four types of response time:
  1. first time renting (ftr) of the service—$respT(ftr)$,
  2. upgrading the service$(upServ)$ by adding additional accounts $(addAcc)$—$respT(upServ, addAcc)$
  3. upgrading the product $(upProd)$—$respT(upServ, upProd)$, and
  4. the service usage $(useServ)$, such as for saving a document (the value of each type of response time is different and predefined in the SLA).
- Penalty Conditions: For each SLA violation the SaaS provider needs to pay a penalty, which is based on the delay in the response time to the customer. For each request type there is a different penalty (detailed in the cost model on Section 2.2.2). Penalty rate is the monetary cost incurred to the provider for unit time delay in serving the customer request.

The *infrastructure layer* (Fig. 1) uses VM images to create instances on their physical infrastructure according to

mapping decisions. The following infrastructure layer properties are important for mapping:

- VM types (l): The type of VM image that can be initiated. For instance, there may be three types of VMs: large, medium, and small. The three types of VMs have different capability to serve different numbers of accounts and records since different requests may consume different memory and storage. Therefore, for a particular type of VM, price, and the maximum capabilities are listed in Table 1.
- Service Initiation Time ($iniT$): How long it takes to initialize the service, which includes the VM initiation time and application deployment and installation time.
- Service Processing Time ($procT$): It is defined as the time taken to process an operation of SaaS service. For example, how long it takes to generate a report, or save a transaction record.
- VM Price ($VMPrice$): How much it costs for the SaaS provider to use a VM for the customer request per hour. It includes the physical equipment, power, network and administration cost.
- Data Transfer Time (DTT): How long it takes to transfer one Gb record from one VM to another. This depends on the network bandwidth.

2.1.2 Customers

When customers register on the SaaS provider's portal, their profile information is gathered. In practice, this happens via forms that customers fill during the registration process. To categorize customers, high level information such as company size in range is collected. For example, the number of information workers, who may be the potential users is between 5 to 10. The following items are considered:

- Company Name ($compName$): The legally registered trade name.
- Company Size ($compSize$): The number of information workers (staffs who may use the software service) in the company.
- Company Type ($compType$): The classification of a customer's company based on the number of employees and revenue. Customer companies are categorized into three divisions, i.e., small, medium, and large.
- Future Interest Expression (*futureInterest*): The customer's expected future upgrade requirements. Such as the need for additional user accounts. This allows the SaaS provider to plan for possible offering of discount as it helps them in making resource reservation decisions. The provider's reduced cost due to advance booking is shared with customer by offering them a discounted price. Such practice is quite commonly used by current industries and service providers. Therefore, we believe that this model will work well for Cloud computing.

Moreover, in the service market, there are two types of sales models, which are one off and long term relationships. The entire sales process is based on relationship building and trust [39]. In addition, the application type we provide is enterprise application, which is used as a pay-as-you-go and most of time with the customer repeatedly using the service. For instance, Company Y may need to use the invoice and report services only a few times a month, but they will use these services repeatedly over the long term. Therefore, we focus on the relationship model but not the once off model (e.g. spot pricing).

2.2 Mathematical Models

2.2.1 Customer Profile Model

Credit Level ($creditLevel$): It measures the creditability of a customer, which depends on the value of the company type and credit level factor (Equation (1)).

$$\text{creditLevel} = \text{compTypeValue} \times \sigma \tag{1}$$

The CompTypeValue indicates the company type, which is categorized based on the range of company size. In practice, the company size can be verified during the registration identity and security verification process. The CompTypeValue for small, medium and large company types are 1, 2, and 3 respectively. The reason we use the values 1, 2, and 3 rather than say 10, 20, 30 or other sets of values, because the trend of other value sets are found to be the same during the evaluation. The company type is considered when calculating the credit level, because having larger companies as customers adds more value to the SaaS provider's market share. The credit level factor (σ) is determined by the customer's historical upgrade requests and the actual upgrade action. The actual upgrade is a boolean value. If an actual upgrade happened, the actual upgrade is true, and otherwise it is false. The value of actual upgrade (actualUpgradeValue) is the actual value, such as number of account, that service upgrades requested. The credit level factor (σ) is the ratio of the actualUpgradeValue and futureInterestValue (which can not be 0) (Equation (2)).

$$\sigma = \frac{actualUpgradeValue}{futureInterestValue}. \tag{2}$$

For example, Company Y expresses a future interest to add 2 user accounts before the contract expiry date. In this case the future interest is 'add user accounts' and the value of the future interest (futureInterestValue) is 2. If they do not come back to request more user accounts (the actual upgrade is false, and the actualUpgradeValue is 0), its credit level factor (σ) is 0; but if it adds one user account (the actual upgrade is true, and the actualUpgradeValue is 1), the credit level factor (σ) is $1/2 = 0.5$ (Eq. (2)). If it adds 3 user accounts (the actualUpgradeValue is 3), the credit level factor is $3/2 = 1.5$. If there is no history about previous actions or user does not specify the future interest value, then σ is 0 (in this case the 'future interest value' is not used for new requests). The customers will specify the future interest every time they submit requests.

This model is used to adjust the inaccuracy or ensure information from the customer using the actually verified and historical data. However it is necessary for providers to keep gathering future interest data from customers, since

customers supplied high level ''future'' expectations/ requirements guided in the initial planning and helps resource providers to plan about possible incentives they may offer to their ''high'' value customers.

2.2.2 Cost Model

Let C be the number of customer requests and c indicates a customer request id. At a given time t, a customer submits a service request c to the SaaS provider. The customer specifies a product edition, contract length, and number of accounts after agreeing with the pre-defined SLA clauses (response time). After the SLA establishment, the SaaS provider will reserve the requested software services which are translated at the infrastructure level to match the VM capacity.

Let $Cost$ be the total cost incurred to the SaaS provider to serve all customer requests C and as described in Equation (3). It depends on the VM cost and the penalty cost.

$$Cost = VMCost + PenaltyCost. \tag{3}$$

Let I be the number of initiated VMs, and i indicates the VM id. The VM cost is the total cost for all VMs and is expressed by Equation (4):

$$VMCost = \sum_{i=1}^{I}(VMCost_i) \quad i \in I. \tag{4}$$

The Penalty cost is the total penalty cost for all customer requests C and is expressed by (5):

$$PenatyCost = \sum_{c=1}^{C} PenaltyCost_c \quad c \in C. \tag{5}$$

For each VM i, the VM cost depends on the VM price of type l ($VMPrice_l$), the time slot when the VM is on (s_i), and the time slot when the VM is off (f_i) and the set up time of the VM i (ts_i) and it is expressed by Equation (6):

$$VMCost_i = VMPrice_l \times (f_i - s_i + ts_i) \quad i \in I, l \in L. \tag{6}$$

Let c' be the previous request from the same customer. The time spent on a VM set up is expressed by Equation (7) and it depends on the request type j, VM initiation time $iniT_i$, total data transfer time for c' ($\mathrm{totalDTT}_{c'}$). If j is 'first time rent' then the data transfer time is zero. Only when j is 'service upgrade' and requires data migration, the data transfer time occurs.

$$ts_i = \mathrm{iniT_i} + \mathrm{totalDTT}_{c'} \quad i \in I, l \in L, c' \in C. \tag{7}$$

The total data transfer time depends on the number of accounts ($a_{c'}$) that previously were requested by the same customer, the data records created by previous request c', the storage size per record ($rs_{c'}$) and data transfer time per size ($DTT_{c'}$). N indicates the total number of records and n is the record id.

$$totalDTT_{c'} = a_{c'} \times \sum_{n=1}^{N} rs_{c'} \times DTT_{c'} \quad n \in N, c' \in C. \tag{8}$$

The SLA violation penalty ($Penalty$) model is similar to the models used in the related publications [1], [3], [4] and is modeled as a linear function. The penalty model is shown in (9). The constant factor α is used to make sure the minimum penalty is always greater than 0. β is the penalty rate and td indicates delay time. β is based on the request type, and each type of request incurs the same range of penalty rate. This is a similar model to credit card penalty, in which the late payment for a particular type of card will have the same range of penalty [42].

$$Penalty = \alpha + \beta \times td. \tag{9}$$

The penalty function penalizes the service provider by increasing the cost. According to the penalty model, the penalty cost equation for each customer request c is depicted as follows where the customer request c is of request type j and $\mathrm{td_c}$ indicates the delay time for customer request c.

$$PenaltyCost_c = \alpha + \beta_j \times td_c \quad j \in J, c \in C. \tag{10}$$

The delay time td is the variation between the value of the response time defined in the SLA and the actual experienced response time. There are four situations in which a penalty delay can occur (Error! Reference source not found.). If the request type is 'first time rent', the delay (violation) can occur due to a long service initiation time. If the request type is 'upgrade service', the delay can be caused by adding accounts or upgrading the product edition. Moreover, during the service usage, the delay can be caused by machine performance degradation, which is out of the scope of this paper.

Average performance can be calculated based on a per-user (macroaverage) or per-request (microaverage). Macro-average performance treats all users equally, although some users will be more active and generate more traffic than others. In contrast, microaverage performance emphasizes the requests made by highly active users. Authors [51] claimed that ''we don't always build per-user predictive models. Individual models of behavior tend to be less accurate because they see less data than a global model. Thus for comparison, we will report only per-request average''. In addition, we consider penalties caused by service preparation response time which are once-off activities without moving average.

The service initiation time varies subjected to the physical machine's capability

$$td_c = \begin{cases} iniT_i - respT_j & \text{where } j = \text{first time rent} \\ iniT_i + totalDTT & \\ \quad -respT_j & \text{where } j = \text{upgraded service.} \end{cases} \tag{11}$$

2.3 Mapping of Products to Resources

In our work, the infrastructure layer focuses on the VM and the host level. The mapping between a host and hosted VMs is depicted in Fig. 2. Our VM to physical machine 'Mapping configuration' supports heterogeneous physical machines. Homogeneous physical machines are depicted just for easy comparison and presentation of results.

We use a similar record model as 'Salesforce.com' to restrict each account to create the maximum number of records. This configuration is chosen to avoid/minimize the SLA violations due to service response delay. Because the VM performance can degrade after a certain number of

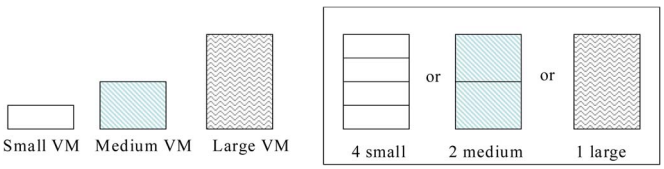

Fig. 2. Mapping between VMs and a Host.

VMs are hosted on the same server due to using shared resources, such as CPU. An example of a mapping strategy between customer requests and VM resources is shown in Table 2.

2.4 Problem Description

Let a SaaS provider have I VMs initiated in a data center, and C is the number of requests currently arriving to the SaaS provider. The SaaS provider charges a fixed service price from customers for an application based on their request parameters. The request parameters include request type (j), product edition (p), contract length (cl), and the number of accounts (a). The SaaS provider has a dual objective, i.e., minimizing the cost and improving CSL. The objective functions and constraint functions are explained below with input parameters and variables:

1. Input Parameters
2. L: Set of VM type.
3. η: Time-slot size.
4. I: Set of VM has been initiated from time 0 to time T. T is divided in slots of size η
5. $\mathrm{VMPrice_l}$: The cost of VM of type l, $l \in \{1, 2, 3\}$.
6. c: The particular request. The parameter of this request includes the number of accounts, when the contract starts, when the contract finishes, which type of request it is, and what type of product it is requesting.
7. C: Set of customer requests received from time 0 to T.
8. β_j: Penalty rate that is associated with request type j.
9. $\mathrm{A_l}$: Maximum number of accounts that can be allocated for VM type l.
10. $\mathrm{a_c}$: The number of accounts requested by request c.
11. $\mathrm{s_c}$: The time slot when this customer request contract started.
12. $\mathrm{f_c}$: The time slot when this customer request contract finished.
13. Variables
14. $\mathrm{y_{il}} = 1$, if VM i is of type l, $\mathrm{otherwise} = 0$.
15. $\mathrm{z_{cj}}$: For request c, $\mathrm{z_{cj}} = 1$, if request c is of request type j.
16. $\mathrm{f_i}$: The time slot when the VM is off.
17. $\mathrm{s_i}$: The time slot when the VM is on.
18. $\mathrm{x_{cit}} = 1$, if request c is served by VM i at time slot t.
19. $\mathrm{td_c}$: The time delayed to serve request c.
20. $\mathrm{ts_i}$: The time spent in setting up the VM i.

Objective Functions In our model we are interested in minimizing the total cost and SLA violations. Consequently, cost minimization can be described by the following function:

$$\mathrm{Minimize\ (Cost)} = \mathrm{VMCost} + \mathrm{PenaltyCost} \tag{12}$$

where

$$\mathrm{VMCost} = \sum_{i=0}^{I}\left(\left(\sum_{l=1}^{3} VMPrice_l y_{il}\right) \times (f_i - s_i + tS_i)\right) \tag{13}$$

$$\mathrm{PenaltyCost} = \sum_{c=1}^{c}\left(\alpha + \sum_{j=1}^{3}(z_{cj} \times \beta_j) td_c\right) \tag{14}$$

In (12), the minimization of cost depends on the VM Cost, and Penalty Cost due to SLA violations. In (13), the VM Cost depends on the type of VM l and the time period VM is on, which is calculated by $(\mathrm{f_i} - \mathrm{s_i})$. VM Cost is the cost of all initiated VM of type l during the time period when the VM is on. In Equation (14), the Penalty Cost depends on the, penalty rate β_j of request type j and time delayed to serve request c (td_c).

The other objective function is to maximize of the CSL by minimizing the SLA violations, which is expressed below:

$$\mathrm{Minimize\ (SLA\ violations)} \tag{15}$$

The number of SLA violations impacts CSL, so we consider minimizing the number of SLA violations as the objective function for maximizing CSL.

Constraints: The SaaS provider needs to ensure that the customer requested product edition, and the number of accounts are allocated before a threshold time (refer to Table 1) to minimize the penalty delay. To this end, we define the following set of constraint functions:

$$\sum_{c=0}^{c} x_{cit} a_c \leqslant \sum_{l=1}^{3} A_l y_{il} \tag{16}$$

$$s_i = \min_{\forall c}\{x_{cit} s_c\} \tag{17}$$

$$\mathrm{f_i} = \max_{\forall c}\{x_{cit} s_c\} \tag{18}$$

$$0 \leq a_c \leq \sum_{i=0}^{I}\left(\sum_{l=1}^{3} x_{cit} y_{il} A_l\right) \tag{19}$$

TABLE 2
The Summary of Mapping Between Requests and Resources

VM Type	VM Capacity and Price	Product Edition	Max Account #	Min Account #
Small	1 CPU Unit, 2Gb RAM, 160 G Disk $0.12 per hour	Standard	M	1
Medium	2 CPU Unit, 4Gb RAM, 850 G Disk $0.48 per hour	Standard, Professional	2m	m+1
Large	4 CPU Unit, 8Gb RAM, 1690 G Disk $0.96 per hour	Standard, Professional, Enterprise	10m	2m+1

```
Base Algorithm Pseudo-code for BestFit
Input        request c with QoS parameters
Output       Boolean
Functions    FirstTimeRent (), Upgrade ()
First Time Rent (c)
1   Let p be the product edition and a_c be the number of accounts required by request c
2   Let L be type of VM which can serve c after applying mapping strategy.
3   Foreach VM i of type 'l' from 'L' to 'Large'
    {  //get list of VMs of type l which can serve the request 'c'
4       Let vmList=GetVMlist(l, p , a_c)
5       If (vmList is empty)
6           continue;
7       Else
        {
8           Allocate capacity of VM_min with minimum available space in vmList to request c
9           update the available capacity of VM_min to (VM_min's available capacity – a_c)
10          break;
        }
    }
11  If(request c is still not served)
    {
12      Initiate a new VM of type L and deploy the product type p on the VM
13      Allocate capacity of the new VM to request c
14      update the available capacity of the new VM to (available capacity – a_c)
    }
Upgrade(c)
1   If (upgrade type is 'add account')
    {
2       Get VM_il which is processing the previous request from the same customer as c
3       If ( VM_il has enough space to serve request c and can guarantee SLA objectives of existing requests)
        {
4           Process request c using VM_il
        }
5       Else
        {
6           Let a_c' be the number of account that are already rented by the customer.
7           Let new a_c be the number of more accounts requested by the customer
8           Using similar process as of the function First Time Rent(c) search a newVM_il which can serve request with
            (a_c'+new a_c) accounts
9           Transfer data from VM_il to newVM_il
10          Release the space in old VM_il
        }
11  If (upgrade type is'upgrade service')
    {
12      get the VM_il which processed the previous request from the same customer as c
13      Using similar process as of the function First Time Rent (c) search a newVM_il which can serve the request
14      Transfer data from VM_il to newVM_il
15      Release the space in old VM_il
    }
```

The Equation (16) restricts the number of accounts requested by all customers on VM i which should be within the maximum capability of the VM of type l (the VM capability is listed in Table 2). In Equation (17), s_i represents the minimum time when customer contract started. In Equation (18), f_i represents the max time when customer contract finished. In Equation (19), the number of accounts (a_c) should be less than or equal to the maximum capability of the VM of type l, which is serving the customer request c.

The objective functions (12) and (15) of the SLA based resource provision problem are to minimize cost and SLA violations for a SaaS provider. The constraints ensure that the customer requirements of an application are met. However, it is difficult to allocate the exact number of accounts to a VM to avoid space wastage within the response time, because customer requests have different parameters, require different types of VMs, and have dynamic arrival rates [43]. Moreover, this problem maps to the 2-dimensional bin-packing problem which is NP-hard [44] (see Appendix A for the proof), hence we propose various algorithms to heuristically approximate the optimum.

3 Resource Provisioning Algorithms

As discussed on the provider side, the main objective of our work is to minimize cost and SLA violations using resource provisioning strategies. We use the best algorithm $(ProfminVMMinAvaiSpace)$ proposed in our previous paper [22] as a benchmark algorithm (renamed to $BestFit$) and propose two new algorithms: $BFResvResource$ and $BFReschedReq$, which consider customer profiles and provider KPI criteria.

3.1 Base Algorithm: Maximizing the Profit by Minimizing the Cost by Sharing the Minimim Available Space VMs (BestFit)

A SaaS provider can maximize its profit by minimizing the resource cost, which depends on the number and type of

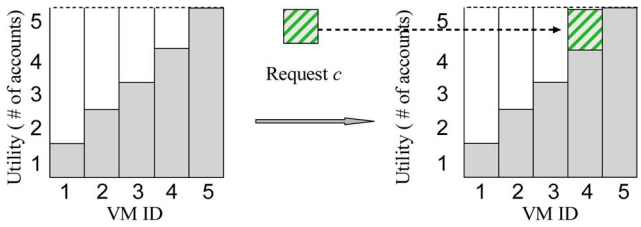

Fig. 3. Best Fit Strategy.

initiated VMs. Therefore, this algorithm is designed to minimize the number of VMs by utilizing the same already initiated one for serving other user requests as well. The algorithm avoids SLA violations of existing requests by not allocating new request to the initiated VM if the new request can cause an SLA violation to existing customers.

The strategy of this algorithm is illustrated in Fig. 3, where the gray space indicates unavailable space, x axis indicates the id of VM, which has the same VM type and is deployed with the same type of product as customer c requested; y axis indicates the number of accounts a VM can hold.

Customer request c is the input of the algorithm, which includes the request type, product edition, and the number of accounts. The algorithm involves two main request types: a) first time rent and b) upgrade service.

If the request type is 'first time rent', the algorithm gets the VM type L using a mapping table similar to Table 2 (Line 1). Then, it checks and gets the list of all initiated VMs of type L (Line 2) that can serve the request 'c' (Line 4). If there is no such initiated VM, it will find space in other types of VMs which are larger in size (Line 5-6). Otherwise, the request c is assigned to the VM from 'vmList' that has minimum available space (Line 8). The available capacity of VM_{min} is updated (Line 9-10) (it is illustrated in Fig. 3). If there is no initiated VM, which can serve the request, then it initiates a new VM according to the mapping strategy and deploys the requested product on this VM (Line 13).

If the request type is 'upgrade', then it checks the type of upgrade. If upgrade type is 'add account', the algorithm gets the id (i) and type (l) of VM, which has placed the previous request from the same customer as c' (Line 2). If VM_{il} has enough space to place the new request c, the algorithm schedules c to VM_{il} (Line 3, 4). Otherwise, the algorithm searches for a $\mathrm{newVM_{il}}$ using a similar way as given in First Time Rent (Line 6-8).Then, the algorithm transfers data stored on the old VM to the new VM and releases space on the old VM (Line 9, 10). On the other hand, if a customer requests an upgrade to a more advanced product edition, the new request is placed to a suitable VM by using the First Time Rent() function, and then the customer's old data is migrated to the new VM and the space occupied by the old request on the old VM is released (Lines 11-15). The time complexity of this algorithm is $O(IC + I)$, where I represents the total number of VMs and C represents the total number of requests.

The ''$BestFit$'' algorithm minimizes the number of initiated VMs to minimize cost. However, the disadvantage is that it can increase the cost in some cases due to delay penalties. For example, when a new customer requests to add more accounts on the VM which has been fully occupied by other requests, initiating a new VM may be more expensive than the delay penalty.

3.2 Proposed Algorithms

- Minimizing the cost by minimizing the penalty cost through resource provisioning based on the customer's credit level $(BFResvResource)$.
- Minimizing the cost by rescheduling the existing requests $(BFReschedReq)$.

3.2.1 *Algorithm 1: Minimizing the cost by Minimizing the Penalty Cost Through Resource Provisioning Based on the Customer's Credit Level (BFResvResource)*

The base algorithm can cause upgrade penalties in the situations when a customer requests to add more accounts and the available space is filled by other requests, because this could trigger the initialization of a new VM. To optimize the cost caused by adding new accounts, *Algorithm 1* provisions more resources than requested based on the customer's credit level (which is driven by customer's actual requirements, the credit level is 0 when the request type is new). When a request's credit level is greater than the provider's expected value, more resources will be provisioned to minimize the time spent on adding user accounts. The algorithm is designed to minimize penalty cost due to the addition of new accounts to the system by reserving resources according to the customer requirements (Line 11). Penalty cost is caused by SLA violations; therefore the reduction of penalty cost will automatically reduce SLA violations. The algorithm also reserves resources according to the historical record and customer estimate to reduce VM cost. Therefore, the total cost (based on VM cost and penalty cost) are minimized.

The customers may be unsure about their future interest, so we design two types of reservation strategies (dynamic and fixed) to figure out how much resources should be reserved. Dynamic reservation $(dynamicR)$ strategy reserves resources for customer request c depending on its credit level $(creditLevel_c)$, the number of accounts $(a_c(futureInterest))$ specified in the future interest and provider's expected value for credit level (its value is '1' in the experiments) using Equation (20). Fixed reservation strategy uses a fixed percentage (e.g., 20 percent) customer specified future interest value instead of credit level.

$$dynamicR = \begin{cases} creditLevel_c \\ \times \mathrm{a_c}\ (futureInterest), & \text{if } creditLevel_c \\ & \geq \text{provider expected value} \\ 0, & \text{otherwise.} \end{cases} \tag{20}$$

The $ReservationStrategy$ is depicted in Fig. 4 (The pattern with horizontal line indicates the reserved resources for the same customer; gray space, x axis and y axis are the same as Fig. 3). The other lines are the same as those in the base algorithm. The time complexity of this

Algorithm 1. Pseudo-code for *BFResvResource*
Input: request c with QoS parameters
Output: Boolean
Functions: FirstTimeRent (), Upgrade ()

```
First Time Rent (c)
1   Let p be the product type and a_c be the number of accounts required by request c
2   Let L be type of VM which can serve c after applying mapping strategy.
3   Foreach VM i of type 'l' from 'L' to 'Large'
    {
4       Let vmList=GetVMlist(l, p , a_c)//get list of VMs of type l which can serve request 'c'
5       If (vmList is empty)
6           continue;
7       Else
        {
8           Allocate capacity of VM_min with minimum available space in vmList to request 'c'
9           CreditLevel = getCreditLevel(Profile Information)
            //get the credit level for request 'c'
10          If (CreditLevel ≥ Threshold)
11              update the available capacity of VM_min to (VM_min's available capacity – a_c(futureInterest))
12          Else
13              update the available capacity of VM_min to (VM_min's available capacity – a_c)
14          break;
        }
    }
15  If (request c is still not served)
    {
16      Initiate a new VM of type L and deploy the product type p on the VM
17      Allocate capacity of the new VM to request c
18      update the available capacity of the new VM to (available capacity – a_c)
    }
Upgrade(c)
1   If (upgrade type is 'add account')
    {
2       Get VM_il which is processing the previous request from the same customer c
3       If ( VM_il has enough space to serve request c and can guarantee SLA objectives of existing requests)
        {
4           Process request c using VM_il
        }
5       Else
        {
6           Let a_c' be the number of account that are already rented by the customer.
7           Let new a_c be the number of more accounts requested by the customer
8           Using similar process as of the function First Time Rent (c) search a newVM_il which can serve request with (a_c'+new a_c)
            accounts
9           Transfer data from VM_il to newVM_il
10          Release the space in old VM_il
        }
    }
11  If (upgrade type is 'upgrade service')
    {
12      get the VM_il which processed the previous request from the same customer c
13      Using similar process as of the function First Time Rent (c) search a newVM_il which can serve the request
14      Transfer data from VM_il to newVM_il
15      Release the space in old VM_il
    }
```

algorithm is $O(IC + I)$ where I indicates the total number of VMs and C indicates the total number of requests.

3.2.2 Algorithm 2: Minimizing the Cost by Rescheduling Existing Requests. (BFReschedReq)

Algorithm 1 prevents the penalties caused by adding accounts but does not prevent penalties caused by upgrading the product edition. Algorithm 2 further minimizes the product edition upgrade penalty by rescheduling accepted requests, which leads to a reduction of SLA violations and total cost (Line 11-26).

The strategy of this algorithm is depicted in Fig. 5 (The pattern with horizontal line indicates the reserved resources for the same customer; gray space, x axis and y axis are the same as Fig. 3). The time complexity of this algorithm is $O(IC + I^2)$ where I indicates the total number of VMs and C indicates the total number of requests.

This algorithm is designed in a way that all VMs are deployed with the full software package to reduce the resource discovery and content migration time for rescheduling accepted requests. If the request type of c is 'service upgrade', the algorithm checks the available space of VM_i which has served the previous request c'. If the available space of VM_i is less than the c required and there is an

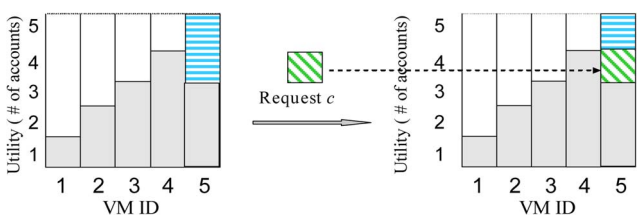

Fig. 4. The Reservation Strategy.

```
Algorithm 2. Pseudo-code for BFReschedReq
Input      | request c with QoS parameters
Output     | Boolean
Functions: | FirstTimeRent (), Upgrade ()
First Time Rent (c)
1    Let p be the product type and a_c be the number of accounts required by request 'c'
4    Let vmList=GetVMlist(p , a_c)//get list of VMs of which can serve request 'c'
5    If (vmList is empty){
8      Allocate capacity of VM_min with minimum available space in vmList to request 'c'
9      CreditLevel = getCreditLevel(Profile Information)
       //get the credit level for request 'c'
10     If (CreditLevel ≥ Threshold )
11       update the available capacity of VM_min to (VM_min's available capacity – a_c(futureInterest))
12     Else
13       update the available capacity of VM_min to (VM_min's available capacity – a_c)
       }
14   Else
       {
15     Initiate a new VM of type L and deploy the product type p on the VM
16     Allocate capacity of the new VM to request c
17     Update the available capacity of the new VM to (available capacity – a_c)
       }
Upgrade(c){
1    If (upgrade type is 'add account')
     {
2      Get VM_il which is processing the previous request from the same customer as c
3      If ( VM_il has enough space to serve request c and can guarantee SLA objectives of existing requests)
       {
4        Process request c using VM_il
       }
5      Else
       {
6        Let a_c' be the number of account that are already rented by the customer.
7        Let new a_c be the number of more accounts requested by the customer
8        Using similar process as of the function First Time Rent (c) search a newVM_il which can serve request with ( a_c'+newa_c)
         accounts
9        Transfer data from VM_il to newVM_il
10       Release the space in old VM_il
       }
     }
11   If (upgrade type is 'upgrade service')
     {
12     get the VM_il which processed the previous request from the same customer as c
13     If ( the available space of VM_il is less than request c required in VM_il ) {
15       If ( migrating c'generates minimum penalty cost || after trying to migrate all requests, available space in
16     VM_il is still less than request c required) {
17         Find or initiate the VM where new and previous requests generate minimum penalty cost
18         Migrate c' and assign c to the VM found or initiated in last step.
19         Transfer all the data to this VM.
         }
20       Else {
21         Find or initiate the VM where migrating other requests generate minimum penalty cost
22         Migrate these requests to the VMs found or initiated in last step.
23         Transfer all the data to this VM.
         }
24       Release the space in old VM_il
       }
25     Else {
26       Allocate c to VM_il;
       }
     }
```

existing request c_e, which causes a lower (or zero) penalty than the current request c, then request c is scheduled on VM_i and the c_e is migrated to another available and capable VM (Upgrade (c)). The request c_e is rescheduled to the cheapest VM. The rest of the lines are the same as those in *Algorithm 1* except that *Algorithm 2* does not differentiate VM types, because all VMs are deployed with the full package. When the customer requests more accounts than the reserved fixed percentage for upgrade, the upgrade function will take care of the exception (Lines 13-26). Briefly, the algorithm checks if the current VM has enough available resources to fit the extra accounts. If yes, the extra accounts will be allocated to the same VM. If no, we will search for the same type of VM with minimum available but enough capability. If there is no suitable VM, *Algorithm 2* need to check if a new VM can be initiated. This may require content migration and incurs penalty cost.

3.3 Lower Bound

Due to the NP hardness of the SLA-based resource provisioning problem described in the system model section, it is difficult to find the optimal solution in polynomial time. Thus, to estimate the performance of our algorithms, we present a lower bound for the cost. The lower bound is derived from the scenario when we can get the minimum cost in case all requests are allocated to the

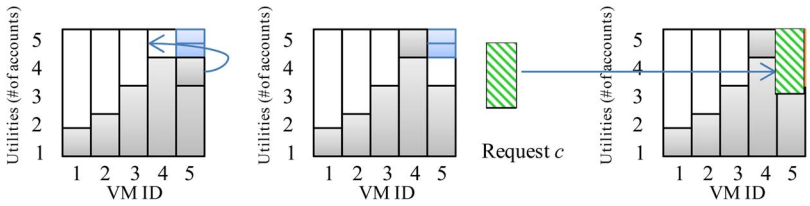

Fig. 5. The Reschedule Strategy.

VM to minimize the VM space wastage, penalty cost and number of SLA violations.

The constraint of the request and VM mapping relationship depends on the number of accounts, product edition, and request type. For the sole purpose of deriving the lower bound, we relax these constraints to minimize the VM space wastage and penalty cost by initiating the large VM to deploy and install the enterprise edition on them. Take the product edition as an example, when the type of the old VM is small, but the customer requests to upgrade product edition to enterprise, which requires the VM of type large but the existing large VMs may do not have enough space for the new request, which causes the penalty. Because all VMs have the same capability, when one VM does not have enough space, we can allocate some accounts to other VMs to minimize VM space wastage. In addition, to relax the dynamic request constraint, the incoming customer requests are known in advance. This forms the ideal lower bound scenario, where all incoming applications are known in advance without any request constraint. c denotes the individual customer request and C denotes the total number of customer requests arrived at time t. a_c denote the number of accounts requested by customer request c. The maximum number of accounts can be accepted by the large VM is defined as M. According to (3), the equation for lower bound is expressed by:

$$\begin{aligned}&\text{Minimize (Cost)}\\&\quad = \text{VMCost} + \text{PenaltyCost; Where PenaltyCost} = 0\end{aligned} \tag{21}$$

$$\begin{aligned}&\text{PerUnitTimeVMCost}_{\text{large}}\\&\quad = \text{VMPrice}_{\text{large}} \times \text{Min}(\text{VM}_{\text{large}}) = \text{VMPrice}_{\text{large}}\\&\qquad \times \frac{\sum_{c=1}^{C} a_c}{M(VM_{large})}.\end{aligned} \tag{22}$$

However, this lower bound solution is the ideal solution, whereas in real dynamic and constraint Cloud environment we cannot achieve the lower bound but can optimize proposed algorithms to be as close as possible to the lower bound. The reason for initiating the large VM to minimize the total cost is proved in Appendix B.

4 Performance Evaluation

We present the performance results obtained from an extensive set of experiments comparing the proposed algorithms with the best algorithm introduced in our previous paper [22]. We discuss the experiment methodology along with performance metrics and detailed QoS parameters. Our analysis of results shows the impact of 1) reservation strategies and 2) QoS parameters: customer's QoS parameters (request arrival rate, proportion of upgrade requests, and credit level) and SaaS provider's parameters (service initiation time and penalty rate).

4.1 Experimental Methodology

We used CloudSim Toolkit [11] to model and simulate the proposed algorithms for resource provisioning. We simulated a data center with 500 physical machines whose configuration resembles are Amazon EC2 large image. A number of VMs of different types that are mapped to a physical machine is shown in Fig. 2. Configuration details of three different types of VMs (small, medium and large) are given in Table 2. The bandwidth of the network connecting physical machine is 10 Gb. The general scheduling policy is time shared scheduling. We have extended the existing Cloud environment and added our algorithm for SLA-based resource provisioning. We model the execution time (i.e. service processing time) based on what we measured from dynamic CRM 4.0 system on a VM with Windows Server 2008R2 OS and 10Gb bandwidth over 2 weekdays and a weekend. For an operation of 303 items records, the mean time for query response time was 2.0 second with a standard deviation of 0.2 second.

We observe the performance of the proposed algorithms by considering performance criteria from both customers' and SaaS providers' perspectives. From customers' perspective, CSL improvement is considered as reducing SLAs violations (from provider's perspectives this is KPI Assurance) and improving service quality (from provider's perspectives this is KPI Performance) in the experiment section. Although in the proposed algorithms only minimization of SLA violations is considered. The number of SLA violations is defined as the number of requests which experience slower response time than the specified in the SLA. Service Quality Improvement (SQI) for an algorithm in the system model is defined as how much faster the actual response time $respT\ (actual)$ than the SLA pre-defined response time $respT\ (SLA)$.

$$\text{SQI} = respT(SLA) - respT(actual). \tag{23}$$

In experiments, Service quality improvement (ServiceQualityImp.) is defined as how much faster the response time of a proposed algorithm is than the base algorithm and is calculated as below:

$$\begin{aligned}&\text{ServiceQualityImp.}\\&= \text{SQI (base algorithm)} - \text{SQI (proposed algorithm)}\end{aligned} \tag{24}$$

From SaaS providers' perspective, how much the total cost is reduced by minimizing the number of VMs is observed.

Therefore, there are four performance measurement metrics: the total cost, number of initiated VMs, percentage of SLA violations, and service quality improvement.

In this paper, experiments are designed from the following three high level considerations:

1. Impact of reservation strategies: The credit level is defined by multiple parameters including 1) company type, which is based on company size 2) customer actual requirements 3) customer expressed future interest. We look into different resource reservation strategies to analyse how dynamic (based on credit level) and fixed reservation strategies impact on performance metrics.
2. Impact of QoS parameters: Which algorithm performs better in which situation by varying arrival rate, proportion of upgrade requests, credit level, service initiation time and penalty rate?
3. Performance Analysis under Uncertainty Future Interest Value: To evaluate the performance of our algorithms in handling the uncertainty in the future interest value.

All the parameters used in the simulation study are given in the following sections.

4.1.1 QoS Parameters

a) Customers' side: From the customers' side, three parameters (request arrival rate, proportion of upgrade requests, and credit level) are varied to evaluate their impact on the performance of our proposed algorithms. Requests arrival rate follows a Poisson distribution as suggested by previous publications [36], [48]. We use a normal distribution $(\text{standard deviation} = (1/2) \times \text{mean})$ to model all parameters, because there is no available workload specifying these parameters.

- Five different types of request arrival rate are used by varying the mean from 200 to 650 simulated customers per second. The probability of a customer to have small, medium and large company type is equal.
- Five different variations in the proportion of upgrade requests are used by varying the mean proportion of upgrade requests from 20 percent to 80 percent.
- Five scenarios vary the proportion of customers having a credit level factor ≥ 1. This proportion is varied from 10 percent to 90 percent ('very low' to 'very high' proportion of companies having high credit level).

b) SaaS providers' side: A SaaS provider offers three product editions (Table 2). Due to unavailability of the public data of the SaaS provider's spending on VMs, we have used the price schema of Amazon EC2 [14] to estimate the cost per hour of using a hosted VM. It is a reasonable assumption, since today many SaaS providers lease resources from IaaS providers rather than maintaining their own resources. Resource price and capabilities, which are used for modeling VMs, are shown in Table 2.

- Five different types of service initiation time (mean value varies from 5 to 15 min) were used in the experiments. The mean of initiation time is calculated by conducting real experiments of 60 samples on Amazon EC2 [14] over four days (2 week days and a weekend) by deploying different editions of products.
- The penalty cost is modelled by (10) and it depends on the request type. The mean of penalty rate (β) varies from \$3 per second (very low) to \$12 per second (very high).

4.2 Results Analysis

We evaluate our proposed algorithms—$BFResvResource$ and $BFReschedReq$ by examining the impact of QoS parameters on the providers' KPIs. For all results, we present the average obtained from 5 experiment runs. In the following sections, we examine various experiments by varying both customers' and SaaS providers' SLA properties to analyze the impact of each parameter. The mean response time which governs SLA violations is set at 5 seconds for 'first time rent' requests, 10 seconds for 'upgrade product' requests and 3 seconds for 'add account' requests.

4.2.1 Impact of reservation strategies

In this set of experiments a dynamic and four fixed (20 percent, 40 percent, 60 percent, and 80 percent) reservation strategies are examined by varying the proportion of high credit level customers, for instance, 20 percent reservation strategies mean reserve 20 percent more space during resource reservation.

In Fig. 6, the variation in credit level (x-axis) indicates the variation in the proportion of customers having high credit level. For instance, the 'very low' credit level indicates that most customers have very low credit level. Fixed (20 percent) reservation strategy costs the least (about 20 percent higher) by utilizing the least number of VMs, but responses slowest (about 60 percent slower) when the credit level is not very low. The dynamic strategy performs the best with respect to the response time but costs the most, because it initiates the largest number of VMs, when the credit level is high.

In regard to the customer satisfaction level, there are two aspects: 1) how many requests experience violations (Fig. 6c), and 2) the service quality improvement (Fig. 6d). In conclusion, during the service type variation experiments, dynamic reservation gives the best service quality improvement, but the fixed reservation saves the most cost. Varying the credit level has the greatest impact on the results, although the overall conclusions are the same as those obtained from the experiments which varied the other parameters, such as upgrade frequency. On the other hand, when the credit level is very low, the dynamic strategy saves the largest amount of cost and incurs the smallest number of SLA violations.

4.2.2 Impact of QoS parameters

I) Impact of arrival rate variation: In this section, we present the performance results of our proposed algorithms in different scenarios. In each experimental scenario, we varied one QoS parameter and set others as constant. For instance, the scenario considered for credit level is 'medium', which indicates the medium proportion of companies with high credit level. The reason for presenting

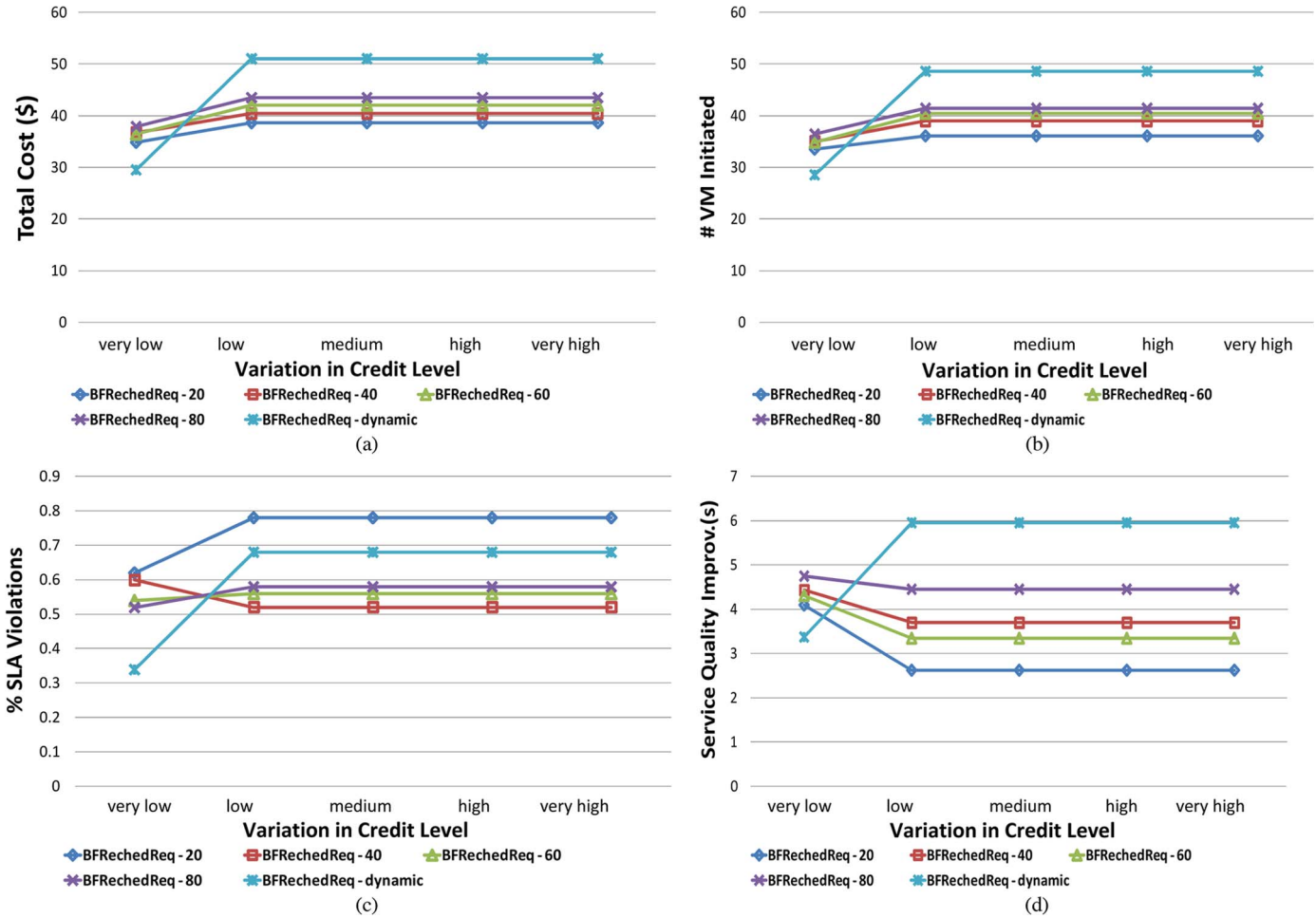

Fig. 6. Impact on reservation strategy during the variation in proportion of customers with high credit level. (a) Total Cost. (b) Number of initiated VMs. (c) Percentage of SLA Violations. (d) Service Quality Improvement.

the 'medium' is to minimize the impact of other factors during the evaluation of reservation strategies. For all experiments, only dynamic reservation strategy is used in algorithms, since it performs best among other evaluated reservation strategies.

The impact of arrival rate on our algorithms is depicted in Fig. 7 with the following parameter settings: 'low' upgrade frequency, 'low' initiation time, and 'medium' for all rest parameters. The lower bound is plotted in line chart. The $BFReschedReq$ is the closest to the lower bound, and it is 18 times and 13 times closer than the $BFResvResource$ and the $base$ algorithm respectively.

On average, the $BFReschedReq$ performs the best by saving about 50 percent of the cost and reducing 60 percent of the SLA violations by using approximately half the number of VMs compared with the base algorithm. As Fig. 7c shows, when the request arrival rate is 'very high', the $BFResvResource$ causes more SLA violations than other algorithms, because when a large number of concurrent requests arrive, they increase the response time for upgrading the services (Fig. 7d). However, the total cost generated by this algorithm is lower than the by the base algorithm due to a lower VM cost. It can be seen from Fig. 7d that $BFReschedReq$ has a smaller improvement in service quality compared with other algorithms, because of the additional time consumed by request rescheduling in transferring data and initiating new VMs. In addition, Fig. 7a and d show that as the service quality improves but costs more. Therefore, during the variation of the arrival rate, the $BFReschedReq$ performs best in respect to the total cost, the number of initiated VMs and causes the least number of SLA violations.

II) Impact of proportion of upgrade requests variation: We investigate the strengths and weaknesses of the algorithms by varying the proportion of upgrade requests from 'very low' to 'very high'. In Fig. 8, 'very low' is when there is no product upgrade but low level of 'add account' upgrade. 'low' is when there is low proportion of both 'product upgrade' and 'add account upgrade'. 'medium', 'high', and 'very high' is when there is 'medium', 'high', and 'very high' proportion of both upgrades respectively. Other parameter settings are: 'very high' for request arrival rate, 'low' for service initiation time, and 'medium' for the rest of parameters. As it can be seen from Fig. 8, the proportion of upgrades increases, the total cost of the base algorithm slightly increases because of more SLA violations while utilizing the similar number of initiated VMs. In contrast, the total cost that is generated by two proposed algorithms decreases, because less number of VMs are initiated by utilizing reserved resources. In the worst case scenario, our proposed algorithms deliver results similar or close to the Best-fit algorithm.

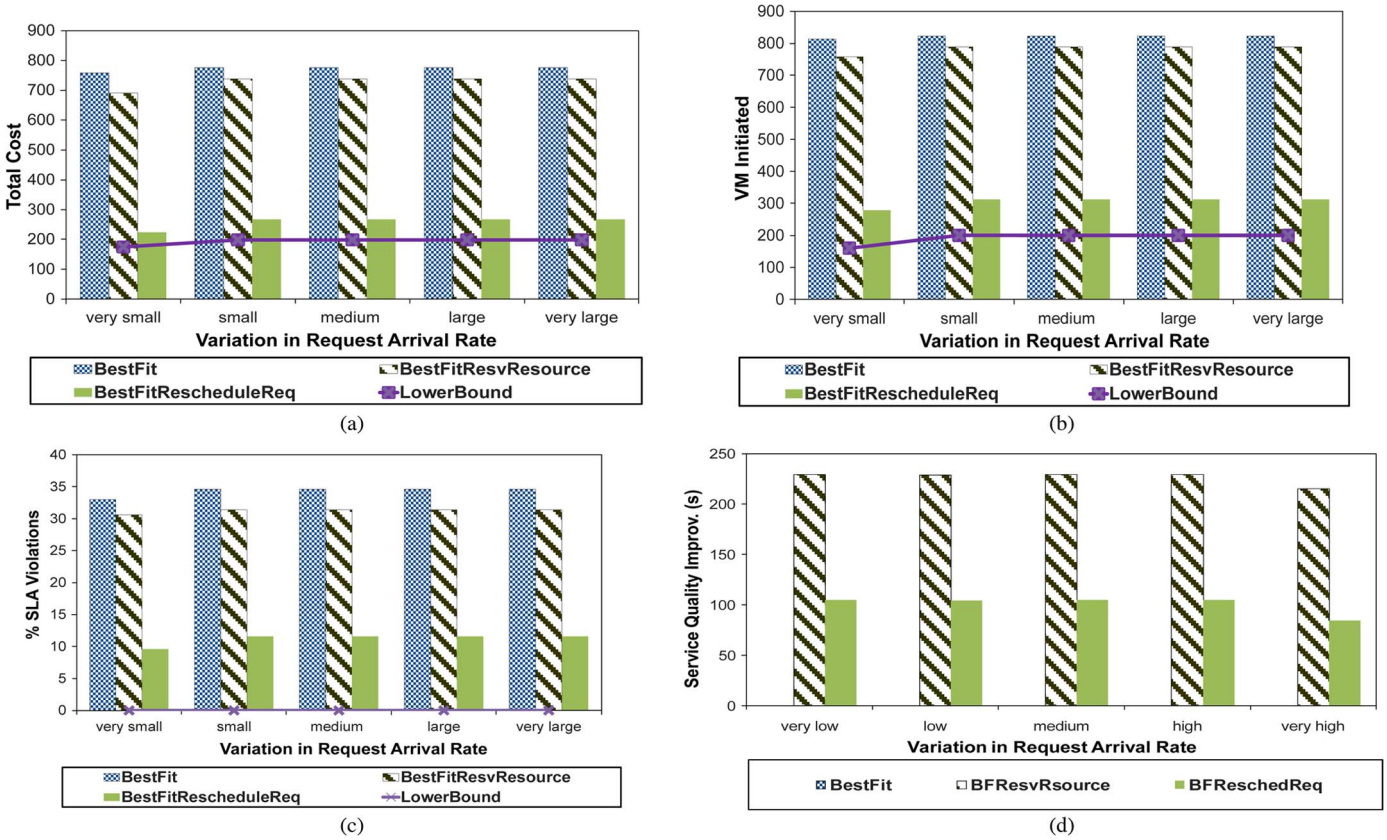

Fig. 7. Impact of request arrival rate variation. (a) Total Cost. (b) Number of initiated VMs. (c) Percentage of SLA Violations. (d) Service Quality Improvement.

When the proportion of upgrade requests is 'very low', $BFResvResource$ saves more cost than the $BFReschedReq$, because $BFReschedReq$ uses large VMs, which cost more than the small and medium VMs. However, when the proportion of upgrade requests varies from 'low' to 'very high', the $BFReschedReq$ saves cost over the $BFResvResource$, because $BFReschedReq$ takes care of product upgrade penalty (SLA violations) and utilizes less VMs to serve an increasing number of product upgrade requests.

To compare with the base algorithm, on average $BFReschedReq$ reduces the cost more than 27 percent when the proportion of upgrade requests varies from 'very low' to 'very high', because it initiates about 30 percent of the number of VMs (Fig. 8b) and SLA violations reduces to about 1 percent (Fig. 8c). The overall trend of SLA violations is increasing (Fig. 8c). Nevertheless, when the upgrade frequency varies from 'low' to 'very high', the $BFReschedReq$ causes more SLA violations than the $BFResvResource$, because the $BFReschedReq$ cannot prevent SLA violations caused by product upgrade.

In regard to the service quality improvement, the $BFReschedReq$ takes more time for rescheduling and the $BFResvResource$ provides better service quality, because the $BFResvResource$ takes about half of the time than that the $BFReschedReq$ takes to respond to the customers' requests (Fig. 8d).

III) Impact of credit level: To investigate the impact of customer profiles, we investigate how the proportion of high credit level customers impacts the performance of our algorithms. In Fig. 9, the variation in credit level (x-axis) indicates the variation in the proportion of customers with high credit level. Parameter settings are: 'very high' value of requests arrival rates, 'very high' value of upgrade proportion, and 'medium' value of all rest parameters. It can be seen from Fig. 9 that there is no influence on the base algorithm, which does not consider customer profiles. However, our proposed algorithms are affected during the variation of proportion of high credit level customers, because our algorithms reserve resources according to the credit level.

When the proportion of high credit level customers varies from 'very low' to 'very high', proposed algorithms generates less cost than the base algorithm by initiating up to 12 percent less number of VMs (Fig. 9b) and violating up to 6 percent less SLA violations (Fig. 9c). This is because the wastage of reserved resources is lower, when the credit level increases. The service quality improvement decreases for both proposed algorithms (Fig. 9d), because it takes longer to serve the same number of requests using fewer VMs.

IV) Impact of service initiation time variation: Fig. 10 shows how service initiation time variation impacts the SaaS provider's total cost. Parameter settings are: 'very high' value of requests arrival rate, and 'medium' value of all rest parameters. When the initiation time varies from 'very short' to 'very long', the trend of the total cost generated by all algorithms increases about 1.5 times, because it causes penalty delays (SLA violations) for new service initiation. The base algorithm is affected more when service initiation time varies from 'long' to 'very long',

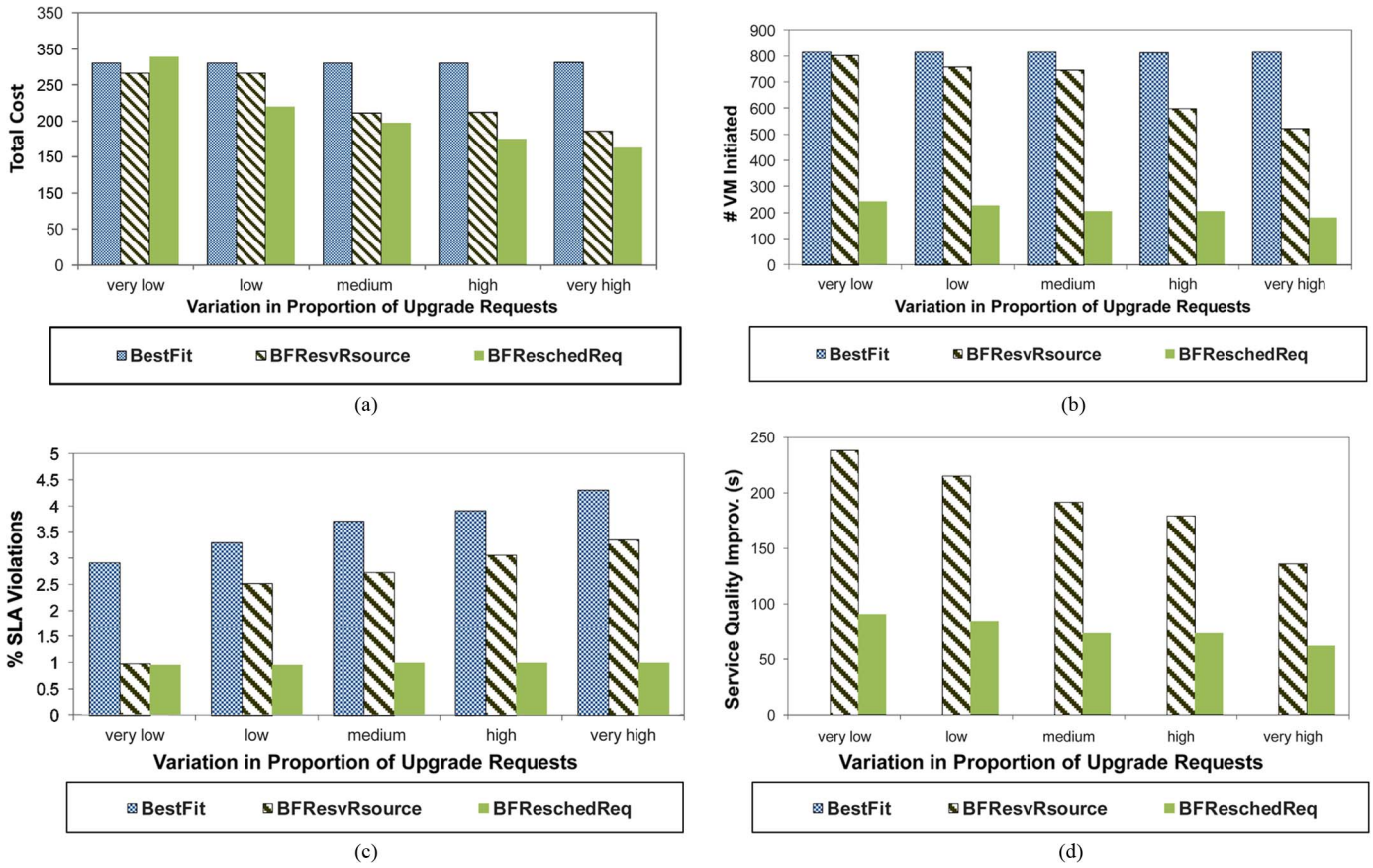

Fig. 8. Impact of proportion of upgrade requests. (a) Total Cost. (b) Number of initiated VMs. (c) Percentage of SLA Violations. (d) Service Quality Improvement.

because it initiates more VMs. The service quality improvement falls down during the enlargement of service initiation time, because the service initiation time includes the time for deploying software services.

V) Impact of penalty rate variation: We investigate how the penalty rate (β) impacts our algorithms. Parameter settings are: 'very high' requests arrival rate, 'low' value of service initiation time, and 'medium' value of all rest parameters. It can be observed from Fig. 11 that all algorithms are affected during the variation of the penalty rate, because requests are scheduled with shared resources. When penalty rate varies from 'very low' to 'very high', the base and the $BFResvResource$ algorithms cost more because of more SLA violations. However, the $BFReschedReq$ saves cost and causes very small number of SLA violations (the maximum percentage is less than 1 percent).

When penalty rate varies from 'medium' to 'very high', the $BFResvResource$ initiates less VMs by using reserved resources, which causes more SLA violations. Because the $BFResvResource$ may delay first time rent requests to serve upgrade requests. In summary, Fig. 11 shows that the $BFReschedReq$ minimizes the total cost, although penalty cost grows during penalty rate variation.

4.2.3 Performance analysis under uncertainty future interest value:

Since customer may be uncertain about their future interest value, they may under-claim or over-claim the value. To evaluate the performance of our algorithms in handling the uncertainty in the future interest value, we carried out two sets of experiments by varying the 1) future interest from 10 percent to 50 percent over-claim (Fig. 12). 2) future interest from 10 percent to 50 percent under-claim (Fig. 13). The base algorithm (BestFit) is not impacted since it does not consider resource reservation.

Fig. 12 shows that during the over-claim of customers' specified future interest value, the total cost (Fig. 12a) increases for both proposed algorithms (up to 10 percent). This is because more VMs are initiated for resource reservation. However, the SLA violations has decreased due to availability of more reserved resources than required.

Fig. 13 shows that during the under- claim of the future interest, the total cost (Fig. 13a) is increasing for both proposed algorithms (up to 2 percent). This is because of more SLA violations, which is due to under allocation of required resources.

The summary of heuristic comparison results regarding to total cost to show on which condition each algorithm can get best and worst results are listed in Table 3.

5 Related Work

Research on market driven resource allocation was started in early 80s [5], [8]. Most market-based resource allocation methods [10] are designed for fixed number of resources [3], [6], [26], [27]. Our work is related to user driven SLA-based economic-oriented resource provision with dynamic number

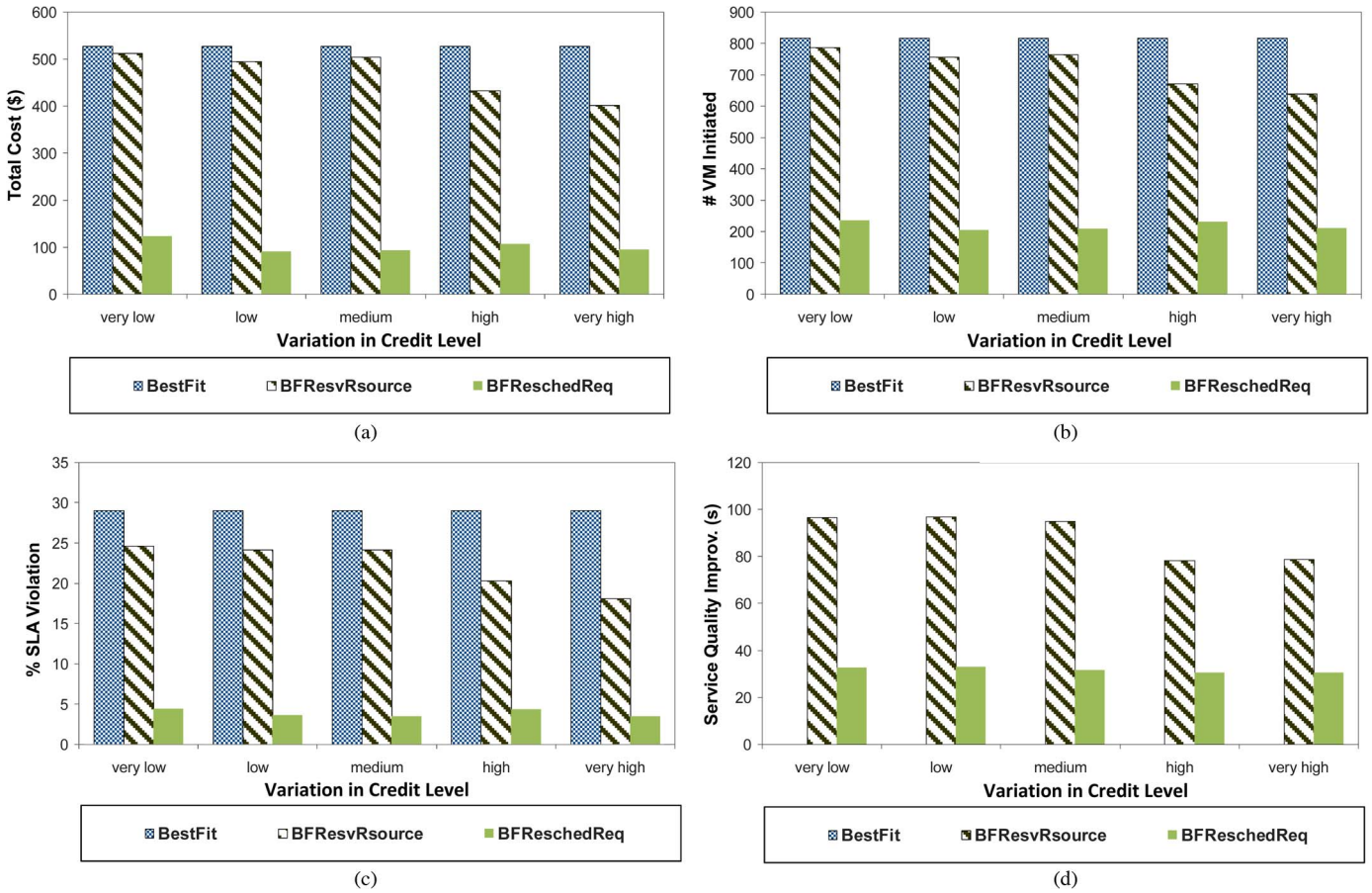

Fig. 9. Impact of variation in proportion of customers having high credit level. (a) Total Cost. (b) Number of initiated VMs. (c) Percentage of SLA Violations. (d) Service Quality Improvement.

of resources. In addition, the resource usage patterns and usage prediction are related areas to our work. The discipline of Web Usage Mining (WUM) has grown rapidly in the past few years, despite the crash of the e-commerce boom of the late 1990s. WUM is the application of data mining techniques to Web clickstream data to extract usage patterns [49]. In the current WUM area, the data has been classified as content, structure, usage and user profile [50]. The first three data categories are related to the usage of Web sites but not the e-commerce transactions. Current three types of usage prediction algorithms, which are history-based, sequence-based and Markov-based algorithms [51], [52] are mainly used in the first three data categories. Thus, in this paper rather than focusing on designing such strategies, we consider user profile and using history-based method for predicting the transaction-based enterprise system usage to calculate the credit level.

In the following sub-sections, we present related publications in Grid and Cloud computing that focus on the area of resource allocation and SLA management.

5.1 Grid

Harnscher *et al.* discussed typical scheduling strategies in computational Grids [24]. They have considered scientific tasks, which run for short term, whereas we consider transaction based applications, which run for long term. Moreover, customer driven scenarios are out of their scope. In addition, the evaluation metrics are different, because they focused on the response time and utilization, while we focus on the cost and the number of SLA violations.

Gomoluch *et al.* proposed market-based resource allocation algorithms for Grid computing [25]. The common points between their and our paper are: firstly, the consideration of state-based and pre-emptive strategies. The state-based strategy indicates all resource allocation based on the current service/system state. The pre-emptive strategy means tasks assigned to a resource, and they are allowed to be migrated to other resources for some advantageous purposes. Secondly, both papers focused on market-based resource allocation. Nevertheless, their work considered independent tasks with input data, deadline as QoS parameters using fixed number of resources. In our case, a customer requests the enterprise applications with multiple QoS parameters using dynamic and flexible resources.

He *et al.* introduced a QoS guided task scheduling algorithm in Grid [38]. The bandwidth was considered as one of the major QoS parameters; and their strategy was based on the earliest completion time, while our paper focuses on minimizing the cost by considering QoS parameters on both customer and provider side.

Reig *et al.* contributed to minimizing the resource consumption for serving requests and executing them within the deadline with a prediction system [12]. Their prediction system enables the scheduling policies to discard the service of a request, if the available resource

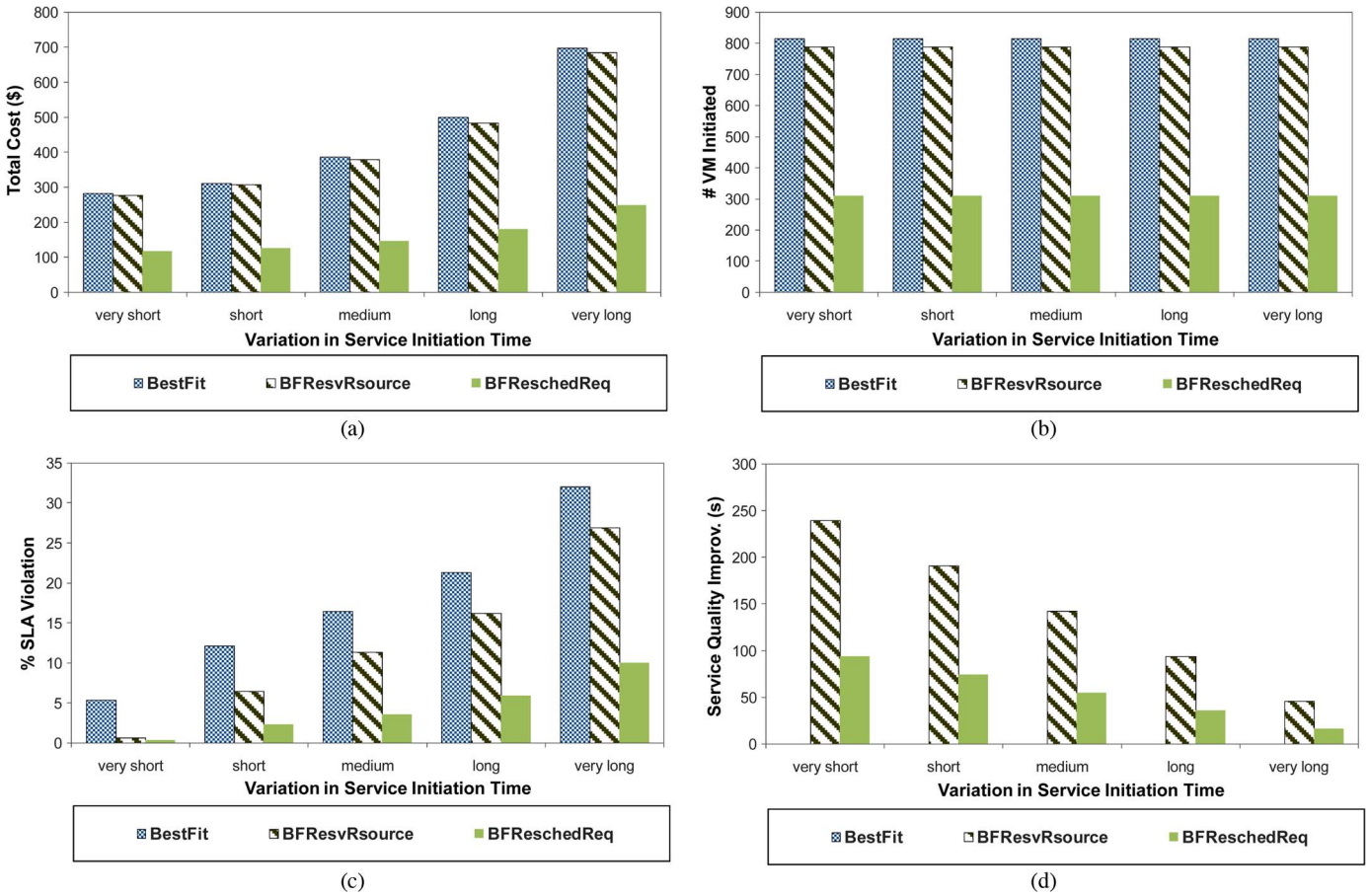

Fig. 10. Impact of service initiation time variation. (a) Total Cost. (b) Number of initiated VMs. (c) Percentage of SLA Violations. (d) Service Quality Improvement.

cannot complete the request within its deadline. However, in our work, we consider the data intensive transaction based application, which run for long term, whereas they considered compute intensive independent application, which are relatively short term. Moreover, the QoS parameters we considered are different from the ones in their work. In addition, our model considers penalty and market oriented targets which do not exist in their work.

Fu *et al.* proposed an SLA-based dynamic scheduling algorithm of distributed resources for streaming [20]. Moreover, Yarmolenko *et al.* evaluated various SLA-based scheduling heuristics on parallel computing resources with two evaluation metrics: resource (number of CPU nodes) utilization and income [21]. Nevertheless, our work focuses on scheduling enterprise applications on VMs in Cloud computing environments (the minimum unit of resources in our work is the number of VMs).

5.2 Cloud

As virtualization is a core technology of Cloud computing, the VM placement has become crucial [31], [32], [33] in the resource management and scheduling, while the virtualization at the operating system (such as, VMware [27]) and storage (such as [28]) level is entering the mainstream. For instance, Grit *et al.* investigated various algorithms for assignment of VMs [31]. Similarly, Van *et al.* proposed the resource provisioning and VM placement [32]. Hermenier *et al.* designed a dynamic consolidation mechanism for homogeneous resources [33]. However, these related publications [31]–[33] did not consider monetary cost or uncertainty of future demand. Bobroff proposed a dynamic heuristic-based VM placement methodology that did not focus on customer-driven scenario to minimize the total cost for SaaS providers [34].

Kimbre *et al.* proposed an allocation algorithm to minimize the number of VM migrations during resource reallocation [29]. Khanna *et al.* pursued the goal to minimize the number of VM migrations and the number of physical machines [30]. In contrast, the objective of our work is to minimize the total cost and number of initiated VMs by considering request migrations instead of VM migrations.

Popovici *et al.* mainly considered QoS parameters on the resource provider's side, such as price and offered load in Cloud computing [6]. Lee *et al.* investigated the profit driven service requests scheduling for dependent tasks without user-driven consideration [2]. In contrast, our work focuses on SLA driven QoS parameters on both user and provider sides; and solves the challenge of assigning dynamically varying customer requests to minimize the cost and number of SLA violations.

Chaisiri *et al.* proposed optimisation of resource provisioning cost in Cloud computing by applying stochastic programming approach in multiple phases [35]. They minimized the cost by considering the uncertainty which is only a part of our objective. In the context of the resource allocation algorithms for enterprise applications, Yang *et al.*

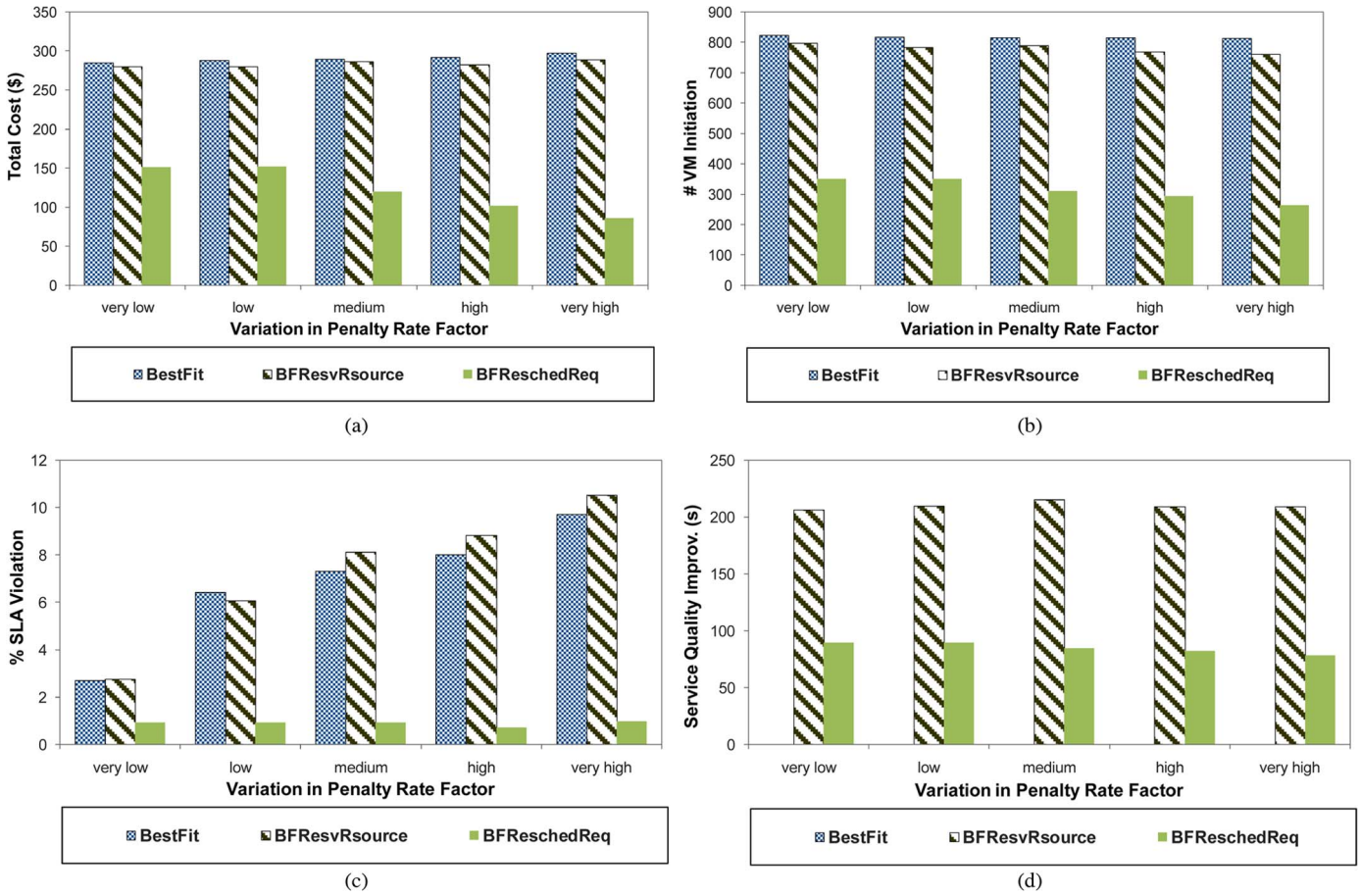

Fig. 11. Impact of penalty rate factor variation. (a) Total Cost. (b) Number of initiated VMs. (c) Percentage of SLA Violations. (d) Service Quality Improvement.

used Genetic Algorithm (GA) in their paper [18]. As GA-based algorithms create a pre-planing schedule, they will not be able to deal with dynamic environment such as Cloud. Therefore, this approach is not suitable for SLA-based resource provisioning in dynamic Cloud computing environments. This paper improves our previous work [22] by proposing two extended algorithms and considering additional QoS parameters such as credit level. We also propose resource provisioning and request migration strategies to optimize the total cost and SLA violations.

In summary, our work is unique in the following ways:

- It manages the CSL based on the customer QoS requirements by minimizing the SLA violations.
- The utility function is time-varying that considers dynamic VM deployment time (service initiation time).
- It considers KPI criteria as a decision making approach for scheduling.
- Scheduling algorithms consider the customer profiles to minimize penalty cost.

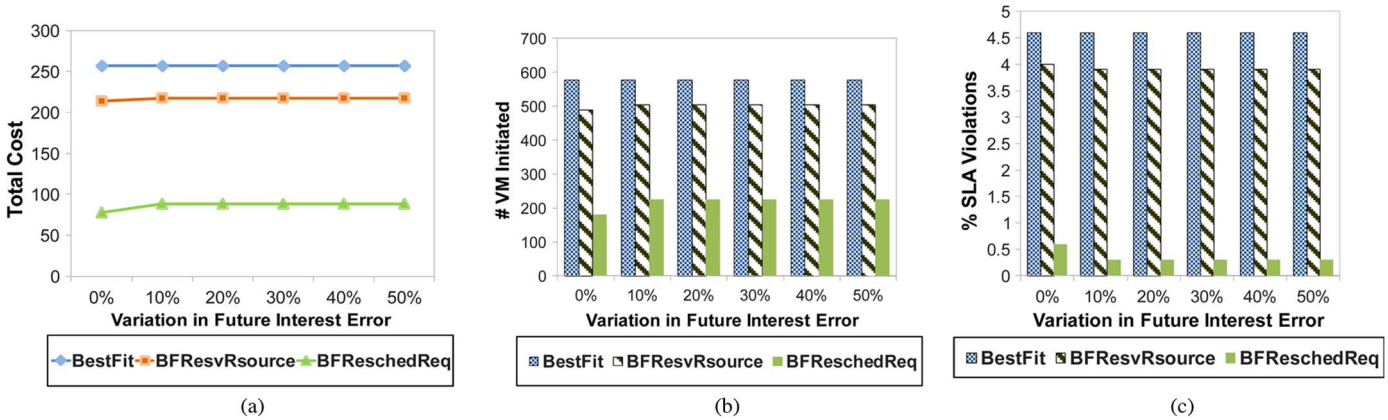

Fig. 12. Impact of future interest error (over-claim). (a) Total Cost. (b) Number of initiated VMs. (c) Percentage of SLA Violations.

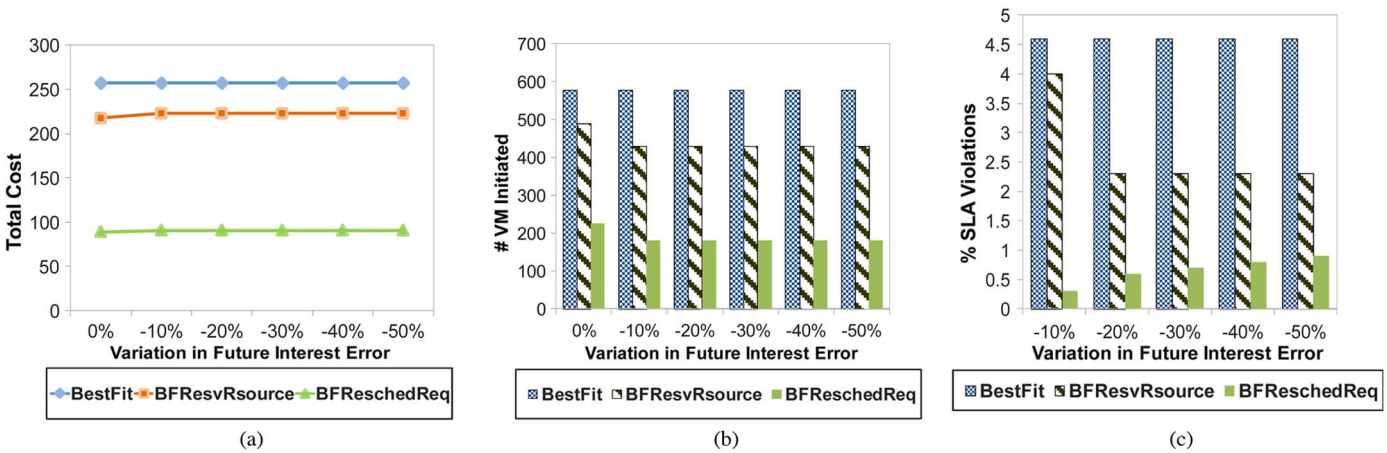

Fig. 13. Impact of future interest error (under-claim). (a) Total Cost. (b) Number of initiated VMs. (c) Percentage of SLA Violations.

- It adapts to dynamic resource pools and consistently evaluates the cost of adding new instances, while most of the previous papers deal with a fixed size of resource pool.

6 Conclusion and Future Directions

In Cloud computing environments, there are primarily three types of on-demand services that are available to customers: Software as a Service (SaaS), Platform as a Service (PaaS) and Infrastructure as a Service (IaaS). This paper focused on resource allocation for SaaS providers with the explicit aim of cost minimization while maximizing CSL by minimizing the number of SLA violations. To achieve this goal, we answered questions raised in the introduction section by considering customer profiles and KPI criteria while using mapping and scheduling mechanisms to deal with the dynamic demands and resource level heterogeneity. We implemented two customer driven algorithms which considered various QoS parameters (such as arrival rate, service initiation time and penalty rate) from both customers' and SaaS providers' perspectives using respectively resource reservation and request rescheduling strategies. In addition, to find out how many resources should be reserved to further optimize the solution, for each QoS parameter, we implemented five sets of reservation strategies (one dynamic and four fixed percentage reservation strategies). The dynamic reservation strategy performed best during the service type variation with respect to the total cost, number of initiated VMs and percentage of SLA violations in general.

The CRM application scenario is a good representative example of many enterprise applications. In addition, the scenario can also be applied to HPC (High Performance Computing) and scientific applications by mapping VM capabilities and QoS requirements. The package upgrade scenario may not be required by them, which

TABLE 3
The Summary of Best and Worst Results (Cost) Comparison

Algorithms	Overall performance				
	Arrival Rate	Proportion of Upgrade Requests	Credit Level	Service Initiation Time	Penalty Rate Factor
BestFit	Best (very small) Worst (very large)	Best (no upgrade) Worst (very high)	No effect	Best (very short) Worst (very long)	Best (very high) Worst (very low)
BFResvResource	Best (very small) Worst (very large)	Best(only add account upgrade) Worst (very high proportion of product upgrade)	Best (very high) Worst(very low)	Best (very short) Worst (very long)	Best (very high) Worst (very low)
BFReschedReq	Best (very small) Worst (very large)	Best (very high proportion of product upgrade) Worst (no product upgrade)	Best (very high) Worst(very low)	Best (very short) Worst (very long)	Best (very high) Worst (very low)

simplifies the scenario compared to enterprise web applications. Therefore, techniques and algorithms proposed in our paper can support a wide range of applications from many domains.

The analysis of our evaluation focused on customers' and SaaS providers' perspectives to maximize various KPI criteria, including the total cost, number of initiated VMs, percentage of SLA violations, and service quality improvement. Simulation results showed that on average, the $BFReschedReq$ results in maximum cost savings and the lowest number of SLA violations compared with the other evaluated algorithms. In general, both proposed algorithms improved service quality to a level higher than that specified in the SLAs and the $BFResvResource$ improved most in regard to the service quality. The lower bound is the ideal solution and the $BFReschedReq$ is the closest to the ideal solution. In addition, in appendix we prove that our problem is NP hardness.

In future, we plan to explore:

1. the SLA negotiation process in Cloud computing environments to improve customer satisfaction levels,
2. resource provisioning for multi-tier applications,
3. considering other pricing strategies such as spot pricing to minimize the cost for service providers,
4. modeling the inaccuracy of customer information and its impact by exploring sophisticated credit level calculation based on the usage pattern and usage prediction technologies, and
5. integration of this work with admission control in Clouds for compute-intensive applications.

ACKNOWLEDGMENT

This work is partially supported by the Australian Research Council (ARC) and CA Technologies. This paper is a substantially extended version of our previous conference paper [22].

REFERENCES

[1] C.S. Yeo and R. Buyya, ''Service Level Agreement Based Allocation of CLUSTER RESOURCES: HANDLING PENALTY to Enhance Utility,'' in *Proc. 7th IEEE Int'l Conf. Cluster*, Bostan, MA, USA, 2005, pp. 1-10.

[2] Y.C. Lee, C. Wang, A.Y. Zomaya, and B.B. Zhou, ''Profit-Driven Service Request Scheduling in Clouds,'' in *Proc. 10th Int'l Symp. CCGrid*, Melbourne, Australia, 2010, pp. 15-24.

[3] O.F. Rana, M. Warnier, T.B. Quillinan, F. Brazier, and D. Cojocarasu, ''Managing Violations in Service Level Agreements,'' in *Proc. 5th Int'l Workshop GenCon*, Gran Canaris, Spain, 2008, pp. 1-10.

[4] D.E. Irwin, L.E. Grit, and J.S. Chase, ''Balancing Risk and Reward in a Market-Based Task Service,'' in *Proc. 13th Int'l Symp. HPDC*, Honolulu, HI, USA, 2004, pp. 160-169.

[5] Y. Yemini, ''Selfish Optimization in Computer Networks Processing,'' in *Proc. 20th IEEE CDC*, San Diego, CA, USA, 1981, pp. 374-379.

[6] I. Popovici and J. Wiles, ''Proitable Services in an Uncertain World,'' in *Proc. 18th SC*, Seattle, WA, USA, 2005, p. 36.

[7] R. Buyya, C.S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, ''Cloud Computing and Emerging IT Platforms: Vision, Hype, and Reality for Delivering Computing as the 5th Utility, Future Generation Computer Systems,'' *Fut. Gener. Comput. Syst.*, vol. 25, no. 6, pp. 599-616, June 2009.

[8] D. Parkhill, *The Challenge of the Computer Utility*. Reading, MA, USA: Addison-Wesley, 1966.

[9] M.A. Vouk, ''Cloud Computing-Issues, Research and Implementation,'' in *Proc. 30th Int'l Conf. ITI*, Dubrovnik, Croatia, 2008, pp. 31-40.

[10] J. Broberg, S. Venugopal, and R. Buyya, ''Market-Oriented Grids and Utility Computing: The State-of-the-Art and Future Directions,'' *J. Grid Comput.*, vol. 3, no. 6, pp. 255-276, Sept. 2008.

[11] R.N. Calheiros, R. Ranjan, A. Beloglazov, C.A.F. De Rose, and R. Buyya, ''CloudSim: A Toolkit for Modeling and Simulation of Cloud Computing Environments and Evaluation of Resource Provisioning Algorithms,'' *Softw., Pract. Exp. (SPE)*, vol. 41, no. 1, pp. 23-50, Jan. 2011.

[12] G. Reig, J. Alonso, and J. Guitart, ''Prediction of Job Resource Requirements for Deadline Schedulers to Manage High-Level SLAs on the Cloud,'' in *Proc. 9th IEEE Int'l Symp. NCA*, Cambridge, MA, USA, 2010, pp. 162-167.

[13] S.K. Garg, R. Buyya, and H.J. Siegel, ''Time and Cost Trade-Off Management for Scheduling Parallel Applications on Utility Grids,'' *Fut. Gener. Comput. Syst.*, vol. 26, no. 8, pp. 1344-1355, Oct. 2010.

[14] C. Vecchiola, X.C. Chu, M. Mattess, and R. Buyya, ''Aneka—Integration of private and public clouds,'' in *Cloud Computing Principles and Paradigms*. Hoboken, NJ, USA: Wiley, 2011, pp. 251-274.

[15] SalesForce.com, Referenced on Dec 6 2010. [Online]. Available: http://www.salesforce.com

[16] Computer Associates Pty Ltd, Referenced on Dec 6 2010. [Online]. Available: http://www.ca.com

[17] Compiere ERP on Cloud, Referenced on Dec 6 2010. [Online]. Available: http://www.compiere.com/

[18] E.F. Yang, Y. Zhang, L. Wu, Y.L. Liu, and S.J. Liu, ''A Hybrid Approach to Placement of Tenants for Service-Based Multi-Tenant SaaS Application,'' in *Proc. 6th IEEE Asia-Pac. Serv. Comput. Conf.*, Jeju Island, Korea, 2011, pp. 124-130.

[19] T. Gad, *Why Traditional Enterprise Software Sales Fail*, Referenced on March 6 2010. [Online]. Available: http://www.sandhill.com/opinion/editorial_print.php?id=307

[20] Y. Fu and A. Vahdat, *SLA Based Distributed Resource Allocation for Streaming Hosting Systems*, Referenced on 6th Dec. 2010. [Online]. Available: http://issg.cs.duke.edu

[21] V. Yarmolenko and R. Sakellariou, ''An Evaluation of Heuristics for SLA Based Parallel Job Scheduling,'' in *Proc. 3rd High Perform. Grid Comput. Workshop*, Rhodes, Greece, 2006, (in conjunction with IPDPS 2006). p. 336.

[22] L. Wu, S.K. Garg, and R. Buyya, ''SLA-Based Resource Allocation for Software as a Service Provider (SaaS) in Cloud Computing Environments,'' in *Proc. 11th IEEE/ACM Int'l Symp. CCGrid*, Los Angeles, CA, USA, 2011, pp. 195-204.

[23] D. Mensce and V. Almeida, *Capacity Planning for Web Performance: Metrics, Models, and Methods*. Upper Sadale River, NJ, USA: Prentice-Hall, 2002.

[24] V. Hamscher, U. Schwiegelshohn, A. Streit, and R. Yahyapour, ''Evaluation of Job-Scheduling Strategies for Grid Computing-Proc. 9th IEEE Int'l Conf. GRID, Tsukuba, Japan, 2000, pp. 191-202.

[25] J. Gomoluch and M. Schroeder, ''Market-based resource allocation for grid computing: A model and simulation,'' in *Proc. 1st Int'l Workshop MGC*, Rio de Janeiro, Brazil, 2003, pp. 211-218.

[26] G. Pacifici, M. Spretzer, and A. Tantawi, ''Performance Management of Cluster BasedWeb Services,'' in *Proc. 11th IEEE/IFIP Symp. Integr. Manage.*, Colorado Springs, CO, USA, 2003, pp. 247-261.

[27] C. Waldspurger, ''Memory Resource Management in VMware ESX Server,'' in *Proc. 5th Symp. Oper. Syst. Design Implementation*, Boston, MA, USA, 2002, pp. 1-15.

[28] G. Alvarez, E. Borowsky, S. Go, T. Romer, R. Becker-Szendy, R. Golding, A. Merchant, M. Spasojevic, A. Veitch, and J. Wilkes, ''Minerva: An Automated Resource Provisioning Tool for Large-Scale Storage Systems,'' *ACM Trans. Comput. Syst.*, vol. 19, no. 4, pp. 483-518, Nov. 2001.

[29] T. Kimbre, B. Schieber, and M. Svirdenko, ''Minimizing Migrations in Fair Multiprocessor Scheuling of Persistent Tasks,'' in *Proc. Annu. ACM-SIAM Symp. Discrete Algorithms*, New Orleans, LA, USA, 2004, pp. 982-991.

[30] G. Khanna, K. Beaty, A. Kochut, and G. Kar, ''Dynamic Application Management to Address SLAs in a Virtulized Server Environment,'' in *Proc. 10th IEEE/IFIP Netw. Oper. Manage. Symp.*, Vancouver, BC, Canada, 2006.

[31] L. Grit, D. Irwin, A. Yumerefendi, and J. Chase, ''Virtual Machine Hosting for Networked Clusters: Building the Foundations for Autonomic Orchestration,'' in *Proc. 2nd IEEE Int'l Workshop Virtualization Technol. Distrib. Comput.*, Tampa, FL, USA, 2006, pp. 1-7.

[32] H.N. Van, F.D. Tran, and J.-M. Menaud, ''SLA-Aware Virtual Resource Management for Cloud Infrastructures,'' in *Proc. 9th IEEE Int'l Conf. Comput. Inf. Technol.*, Xiamen, China, 2009, pp. 357-362.
[33] F. Hermenier, X. Lorca, and J.-M. Menaud, ''Entropy: A Consolidation Manager for Clusters,'' in *Proc. ACM SIGPLAN/ SIGOPS Int'l Conf. VEE*, Washington, DC, USA, 2009, pp. 41-50.
[34] N. Bobroff, A. Kochut, and K. Beaty, ''Dynamic Placement of Virtual Machines for Managing SLA Violations,'' in *Proc. 10th IFIP/IEEE Int'l Symp. IM*, Munich, Germany, 2007, pp. 119-128.
[35] S. Chaisiri, B. Lee, and D. Niyato. (2012, Apr.-June). Optimization of Resource Provisioning Cost in Cloud Computing. *IEEE Trans. Serv. Comput.* [Online]. *5(2)*, pp. 164-177, preprint. Available: http://doi.ieeecomputersociety.org/10.1109
[36] M.L. McManus, M.C. Long, A. Copper, and E. Litavak, ''Queuing Theory Accurately Models the Need for Critical Care Resources,'' *Anesthesiology*, vol. 100, no. 5, pp. 1271-1276, May 2004, Lippincott Williams & Wilkins; ISBN (0003-3022), USA.
[37] R.W. Wolff, ''Poisson arrivals see time averages,'' *Operations Research*, vol. 30, no. 2, pp. 223-231, 1982.
[38] X.S. He, X.H. Sun, and G. Von Laszewski, ''QoS Guided Min-Min Heuristic for Grid Task Scheduling,'' *J. Comput. Sci. Technol.*, vol. 18, no. 4, pp. 442-451, July 2003.
[39] A. Bryant and B. Colledge, ''Trust in electronic commerce business relationships,'' *J. Electron. Commerce Res.*, vol. 3, no. 2, pp. 32-39, 2002.
[40] S. Crago, K. Dunn, P. Eads, L. Hochstein, K. Dong-In, K. Mikyung, D. Modium, K. Sigh, S.J. Woo, and J.P. Walters, ''Heterogeneous Cloud Computing,'' in *Proc. IEEE Int'l Conf. CLUSTER*, Austin, TX, USA, 2011, pp. 378-385.
[41] A. Sumit, J. Driscoll, X. Gabaix, and D. Laibson, ''The Age of Reason: Financial Decisions Over the Life-Cycle With Implications For Regulation,'' in *Proc. Brooking Papers Econom. Activity*, Fall 2009, pp. 51-117.
[42] M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. San Francisco, CA, USA: Freeman, 1979.
[43] Y. Chen, A. Das, W. Qin, A. Sivasubrammaniam, Q. Wang, and N. Gautam, ''Managing Server Energy and Operational Costs in Hosting Centers,'' *ACM Sigmetrics Perform. Eval. Rev.*, vol. 33, no. 1, pp. 303-314, June 2005.
[44] S. Martello and P. Toth, ''An Algorithm for the Generalized Assignment Problem,'' in *Proc. 9th IFROS Conf. Oper. Res*, 1981, pp. 589-603.
[45] Goolgle App Engine, Referenced on June 6 2012. [Online]. Available: http://www.google.com/enterprise/apps/business
[46] Hyper-V, Referenced on June 6 2012. [Online]. Available: http://www.microsoft.com/en-us/server-cloud/hyper-v-server/default.aspx
[47] VMWare, Referenced on June 6 2012. [Online]. Available: http://www.vmware.com/
[48] L. L Wu, S.K. Garg, and R. Buyya, ''SLA-Based Admission Control for a Software-as-a-Service Provider in Cloud Computing Environments,'' *J. Comput. Syst. Sci.*, vol. 78, no. 5, pp. 1280-1299, Sept. 2012.
[49] R. Cooley, ''The Use of Web Structure and Content to Identify Subjectively Interesting Web Usage Patterns,'' *ACM Trans. Internet Technol.*, vol. 3, no. 2, pp. 93-116, May 2003.
[50] J. Srivastava, R. Cooley, M. Deshpande, and P.-N. Tan, ''Web Usage Mining: Discovery and Applications of Usage Patterns From Web Data,'' *SIGKDD Explorations*, vol. 1, no. 2, pp. 12-23, Jan. 2000.
[51] D.D. Brian, ''Learning Web Request Patterns,'' in *Web Dynamics: Adapting to Change in Content, Size, Topology and Use*, A. Poulovassilis and M. Levene, Eds. Berlin, Germany: Wiley, 2004, pp. 435-460.
[52] Z. Su, Q. Yang, Y. Lu, and H. Zhang, ''Whatnext: A Prediction System for Web Requests Using N-Gram Sequence Models,'' in *Proc. First Int'l Conf. Web Inf. Syst. Eng.*Hong Kong, 2000, pp. 214-221.

Linlin Wu is pursuing the PhD degree at the University of Melbourne. She has received a best paper award for her Cloud computing paper published in AINA 2010 conference. Her research interests include resource management, scheduling, utility and Cloud computing, Knowledge Management and Business Intelligence. She holds Project Management and SharePoint Professional Certificates from PMI and Microsoft.

Saurabh Kumar Garg is a Lecturer of Computing and Information System at the University of Tasmania. He is one of the few PhD students who completed in less than three years from the University of Melbourne in 2010. He has published more than 30 papers in highly cited journals and conferences with H-index 18 as an early career researcher. His research interests include resource management and scheduling, Cloud computing, Data analytics and Stream computing. His doctoral thesis focused on devising novel and innovative market-oriented meta-scheduling mechanisms for distributed systems under conditions of concurrent and conflicting resource demand.

Steve Versteeg received the PhD degree in computer science from the University of Melbourne, Australia. He is a Research Staff Member with CA Labs Australia. His current projects are in the areas of cloud computing, software engineering, large scale endpoint emulation, role engineering and insider threat prediction. Dr. Versteeg's doctoral research was in the area of neural simulation. From 2004 until early 2008, he worked at WMind LLC as a Senior Developer and Researcher on an experimental automated futures trading system.

Rajkumar Buyya is Professor of Computer Science and Software Engineering; and Director of the Cloud Computing and Distributed Systems (CLOUDS) Laboratory at the University of Melbourne, Australia. He has authored over 400 publications and four text books. He has authored over 425 publications and four text books including ''Mastering Cloud Computing'' published by McGraw Hill and Elsevier/ Morgan Kaufmann, 2013 for Indian and international markets respectively. He is currently serving as the foundation Editor-in-Chief (EiC) of *IEEE Transactions on Cloud Computing*.