Cost-aware Cloud Service Request Scheduling for SaaS Providers

Zhipiao Liu, Shangguang Wang*, Qibo Sun, Hua Zou, Fangchun Yang
State Key Laboratory of Networking and Switching Technology
Beijing University of Posts and Telecommunications
Beijing 100876, China
{liuzp, sgwang, qbsun, hzou, fcyang}@bupt.edu.cn

∗Corresponding author: sgwang@bupt.edu.cn

Abstract As cloud computing becomes widely deployed, more and more cloud services are offered to end users in a pay-as-you-go manner. Today's increasing number of end user-oriented cloud services are generally operated by SaaS (Software as a Service) providers using rental virtual resources from third-party infrastructure vendors. As far as SaaS providers are concerned, how to process the dynamic user service requests more cost-effectively without any SLA violation is an intractable problem. To deal with this challenge, we first establish a cloud service request model with SLA constraints, and then present a cost-aware service request scheduling approach based on genetic algorithm. According to the personalized features of user requests and the current system load, our approach can not only lease and reuse virtual resources on demand to achieve optimal scheduling of dynamic cloud service requests in reasonable time, but also can minimize the rental cost of the overall infrastructure for maximizing SaaS providers' profits while meeting SLA constraints. The comparison of simulation experiments indicates that our proposed approach outperforms other revenue-aware algorithms in terms of virtual resource utilization, rate of return on investment and operation profit, and provides a cost-effective solution for service request scheduling in cloud computing environments.

Keywords: Cloud computing; Cloud service; Cost; SaaS; Service request scheduling; Virtual machine.

1. Introduction

As a promising computing paradigm, cloud computing has drawn extensive attention from academia and industry in recent years. Cloud computing is formally defined as an IT resource supply model which provides users with configurable computing resources (e.g., servers, storage, applications) over network in the form of services [1,2]. These services are made available on a subscription basis using pay-as-you-use model to cloud users, regardless of their location. Nowadays almost every well-known IT company, including Amazon, Google, IBM and Salesforce, has introduced related cloud services.

Compared with traditional desktop computing, cloud computing presents many advantages, such as better resource utilization, rapid elasticity, higher power conservation and economies of scale, which can save the up-front investment of enterprise information system and reduce the daily operation and maintenance costs significantly in the long run.

With the advancement of cloud computing technologies including virtualization, security, SOA (Service-Oriented Architectures) and high bandwidth network access, it is becoming a trend that large numbers of existing business applications from companies and institutes will be migrated into clouds and deployed as cloud services due to the above-mentioned benefits [3,4]. Therefore, more and more cloud services hosted by cloud service providers (e.g., SaaS providers) will be provided to interested end users, which are deployed on virtual machine (VM) instances rented from one or more third-party infrastructure vendors. Hereafter, the terms cloud service provider and SaaS provider are used interchangeably in the context of this paper.

As a result, a three-tier cloud service provision structure has been formed involving three typical parties: end user, cloud service provider and cloud infrastructure vendors [5, 6]. An end user is the cloud consumer which represents a person or organization that maintains a business relationship with, and requests the cloud service from a business service provider [7]. A cloud service provider such as Force.com is the business service provider which deploys and runs the business applications on a rented cloud infrastructure so that the cloud services are offered to end users through network access. A cloud infrastructure vendor such as Amazon is the entity which provisions virtual resources such as processing, storage, networks, and other fundamental computing resources to clients in a pay-as-you-go manner.

The service provision procedure can be briefly described as follows:

- Above all, the end user browses the service catalog from a SaaS provider and sends the appropriate service request to the cloud service provider.
- The cloud service provider accepts the service request of end user and applies to the underlying cloud infrastructure vendors such as Amazon for virtual resources on demand.
- The cloud infrastructure vendor responds to the resource lease request, and then allocates VM instances to the corresponding SaaS provider for processing the end user request.
- Finally, the SaaS provider charges end user for processing his/her service request and pays the cloud infrastructure vendor for renting VM instances to deploy service capacity.

The involved parties and their interaction can be illustrated in Figure 1.

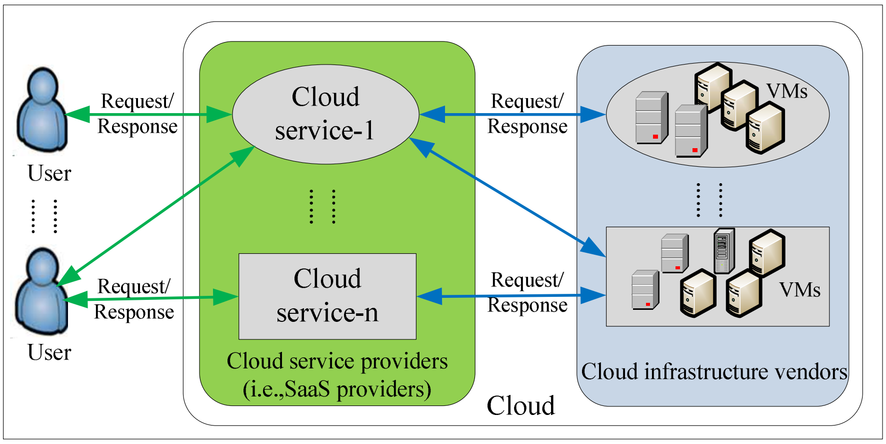

Figure 1 Three-tier cloud service provision structure

In this paper, we only care about the interests of end users and SaaS providers. From the end user's viewpoint, a service request for a business application is always accompanied by SLA (Service Level Agreement) constraints specifying the performance requirements [7]. From the viewpoint of a SaaS provider, the operational goal is to lease as little virtual resource as possible while still ensure that the cloud service is provisioned at the expected service levels to end users. The profits of SaaS providers derive from the margin between the revenue generated from cloud end users and the rental cost of infrastructure.

As can be seen from Figure1, the cloud service provider plays an important role in cloud service provision procedure. For a SaaS provider, how to schedule virtual resources leased from third party infrastructure to process dynamic cloud service requests more cost-effectively without violating the SLA constraints while maximizing operational profit is an intractable problem. On the one hand, service request scheduling strategies in cloud computing environments should balance service performance and the cost of leasing resources to satisfy the objectives of both end users and SaaS providers. On the other hand, it must also recognize and reflect the different options for computing resources (e.g., multiple infrastructure vendors offer many types of virtual machines, each with different capabilities at a different price) [8]. Furthermore, the current pricing model of virtual resources specified by infrastructure vendors should be taken into consideration. All these factors make cost-effective service request scheduling a challenging problem to solve in cloud computing scenario.

In order to deal with this challenge, we stand in the position of cloud service providers and propose an effective solution to achieve optimal cloud service request scheduling. Above all, a cloud service request model with SLA constraints is established. And then, based on the request model, we present a novel optimization scheduling approach, i.e., cost-aware service request scheduling based on genetic algorithm (called CSRSGA). Taking into consideration the divisibility feature of cloud service requests and the elasticity of SLA, CSRSGA intends to maximize the overall infrastructure leasing cost while still ensuring that the service performance can meet SLAs expectation of end user requests. Given the fluctuating service request volume and the huge searching space of virtual resource pool, genetic algorithm is adopted by our approach to improve the efficiency of problem solving and respond to users' requests in reasonable time. In order to verify the effectiveness of our proposed scheduling approach, extensive simulations are conducted based on Amazon EC2 on demand instances. The experiment results show that CSRSGA outperforms other revenue-aware algorithms in terms of virtual resource utilization, rate of return on investment and operation profit.

The main contributions of our work are listed as follows:

- We develop a cloud service request model with SLA constraints based on previous work to identify the main concerns of both cloud consumers and cloud service providers.
- On the basis of the divisible features of the user request and the current system load, we propose an effective service request scheduling approach for maximizing profit by cost and revenue optimization without any SLA violation, and thus reach win-win solution which will help to build a long-term profitable cloud service market.
- As a parallelizable modern intelligent optimization algorithm, genetic algorithm is adopted for achieving optimized request dispatching in reasonable time by incorporating the heterogeneity of virtual resource (e.g., VMs) in terms of their configuration, performance and price.

The rest of the paper is organized as follows: Section 2 introduces the prior work related to service request scheduling; Section 3 presents the cloud service request model with SLA constraints and the revenue function of cloud service providers; Section 4 describes our cloud service request scheduling approach; Section 5 presents the simulation results and comparative analysis; Section 6 concludes the paper and proposes future work.

2. Related Work

Earning profit is the principal driving force for service providers, and SLA is the focus of users' attention. Therefore, much research has been done related to the two themes in distributed computing environment.

In [9], a computational economy driven scheduling system called Libra was presented to support allocation of resources based on the users' quality of service requirements, which offers market-based economy driven service for managing batch jobs on clusters by scheduling CPU time according to user-perceived utility, but pays little attention to system performance. In [10], a sigmoidal utility model was introduced, and several allocation policies for resource allocation on computational grids were proposed. The authors of [11] focus on the profit-based scheduling and admission control policies to address the resource allocation problem from the viewpoint of resource providers, but ignore the user side.

In addition, there have been several recent related efforts in the area of service request scheduling in cloud computing scenarios. The authors of [5] introduce utility theory leveraged from economics, investigate the interaction of service profit and customer satisfaction, but the proposed scheduling algorithms based on resource bid do not respond to end users' requests until the next time interval. Due to the fact that the bid time interval cannot be too short in practice, the long waiting time increases the probability of SLA violation in cloud computing environment, where cloud consumers need to be served immediately, and thus reduces significantly the profits of cloud service providers. In [6], a pricing model using processing-sharing was developed, and two profit-driven scheduling algorithms for composite services in clouds were proposed. In [12], a decentralized economic approach for dynamically adapting the cloud resources of various applications considering the varying workloads or failures was presented. The authors of [13] propose resource allocation algorithms for SaaS providers who want to minimize infrastructure cost and SLA violations. However, the scheduling algorithms proposed in the above three papers cannot completely eliminate the occurrence of SLA violation event.

Our proposed request scheduling approach differs from the prior work mainly in the cloud user request model we introduce, and the optimized scheduling strategy based on the personalized feature of user requests and dynamic resource reuse. Our approach builds and dynamically maintains a virtual resource pool, achieves optimal request scheduling in reasonable time, and thus significantly improves resource utilization and reduces operational cost to increases profits of cloud service providers while meeting end users' performance requirements, which is absent from most previous works in cloud computing environments.

3. User Request Modeling

In order to design a cost-effective cloud service request scheduling algorithm, a reasonable service request model has to be established firstly in order to quantify the critical SLA property constraints.

A SLA is a contract between a service provider and a user, which is a collection of service level requirements that formally specify the promised service performance and the corresponding revenue (or penalty). Generally speaking, SLAs include such predefined properties as response time, user budget, reliability, remedies for performance failures, etc. [7].

In cloud computing environments, SaaS providers need SLAs to regulate user behavior for achieving expected benefits. From a cloud user's point of view, it is also necessary to signing a legal contract covered SLA constraints to specify the technical performance requirements fulfilled by a cloud service provider [7]. Failure to achieve these performance objectives over a period of time binds the SaaS provider to pay a penalty to the cloud user based on the clauses defined in the SLA contract. For example, if the cloud user gets the corrective responsive result within the promised time from the corresponding SaaS provider, then the user arranges payment for the service provisioned accordingly. However, if the user request is not addressed correctly on time, then the SaaS provider will incur penalty. Therefore, SaaS providers always manage to reduce SLA violations to maximize its net profit, that is, the total fees (e.g., revenue) charged by the SaaS provider to its customer minus the cost for renting resource from the infrastructure vendors and the penalties for violating SLA constraints agreed by both parties.

In this paper, we focus on SLA constraints on request processing time (i.e., the time elapsed from accepting a user service request to completion) and cost (i.e., the user's budget for processing this request), which are directly associated with the profit of SaaS providers and cloud consumers. In addition, we assume that every request of end user is subject to corresponding SLA constraints.

According to the above introduction, centered on the two main time and cost constraints, we quantify the other SLA properties related to profit, and then model the user service request $request$ as a five-parameter tuple as follows:

$$request = (budget, t_s, mbdt, deadline, pr) \tag{1}$$

- $budget$: The maximum amount of currency that the user is willing to pay for the request to be completed, i.e., the maximum revenue acquired by the SaaS provider for processing this user request.
- t_s: The standard execution time required to finish the request by a standard VM instance.
- $mbdt$: The maximum processing delay without any penalty incurred by service providers. In order to get the maximum revenue, the SaaS provider should try to complete the request processing before this time point. Otherwise, revenue loss is inevitable for this cloud service provider.
- $deadline$: The processing time upper limit. If the user request is finished after this limit, a SLA violation event occurred. The service provider will compensate the user for failing to meet the deadline of this request. The amount of compensation depends on the delay time, which can be calculated based on the above mentioned two time constraints specified in SLA and the actual processing time.
- pr: Penalty rate. A greater penalty value means that the user requirement in term of request processing time is more demanding. If the actual processing time is greater than the $mbdt$ value of this user request, the reduced revenue should be calculated based on the penalty rate, which establishes a correlation between the request processing time and the revenue of SaaS providers.

For simplicity, we model the SLA violation penalty rate as linear [14], as shown in Figure 2.

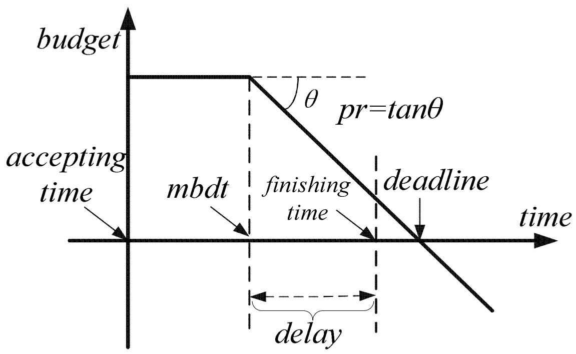

Figure 2 Cloud service request model with SLA constraints

According to the above established cloud service request model, we can easily get the revenue function of service providers.

$$revenue = \begin{cases} budget, & t_a \le mbdt \\ budget - delay * pr, & mbdt < t_a \le deadline \\ -delay * pr, & t_a > deadline \end{cases} \tag{2}$$

- $revenue$: The revenue of the cloud service provider after the user request processing is completed.
- t_a: The actual execution time of this user request.
- $delay$: The execution delay introduced to calculate the incurred penalty of service providers, which can be figured out as follows:

$$delay = \begin{cases} t_a - mbdt, & if\ mbdt < t_a \le deadline \\ t_a - deadline, & if\ t_a > deadline \end{cases} \tag{3}$$

As a result, we can obtain the final revenue of the cloud service provider for processing this user request by formula (2).

4. Our Proposed Cloud Service Request Scheduling Approach

In order to meet diverse market needs, the popular infrastructure vendors such as Amazon and Microsoft offer multiple types of VM instances, which have different configurations, different capacities at different prices. From the viewpoint of SaaS providers, due to the considerable diversity in performance and prices of different types of instances, processing a request on different types of VM instances results in different processing time, and hence different profits. For example, as the leading cloud infrastructure provider, Amazon EC2 currently offers many on-demand VM instance types that differ in computing/memory capacity, OS type, pricing scheme and geographic location for rent on an hourly basis, such as standard VMs (including Small, Large and Extra Large instances) designed for most types of applications, high-CPU VMs for compute intensive services and high-memory VMs for data storage services [5,15].

In this context of resource heterogeneity, processing a user request on different types of VM instances results in different processing time and revenue, and hence different profits because of the considerable discrepancy in performance and prices of different types of instances. It is in the cloud service provider's interests to determine that what types of VM instances and how many instances are leased to minimize infrastructure cost and optimize operational profit. Therefore, it is particularly important to design a cost-aware service request scheduling algorithm for processing highly dynamic user requests in the context of resource heterogeneity. Next, we first introduce a parameter named virtual machine capacity quantity ratio, and then describe our proposed cost-aware cloud service request scheduling approach in detail.

4.1 Capacity Quantity Ratio

To quantify the performance difference of different types of VM instances, we introduce the following conception based on the work of reference [5].

Definition 1 Virtual machine capacity quantity ratio: let rw_i and rw_s denote the request workload that a standard VM instance vm_s and a type i VM instance vm_i can process in a time unit respectively. The virtual machine capacity quantity ratio denoted by qr_i is defined as follows:

$$qr_i = rw_i / rw_s \tag{4}$$

Based on this parameter, the time of processing the same user request on different types of VM instances can be figured out. The qr_i value of various instance types can be determined through profiling and benchmarking.

If the required processing time of certain user request on a standard instance is t_s, then the required processing time on a type i instance can be calculated as follows:

$$t_i = t_s / qr_i \tag{5}$$

where the greater value of qr_i means more powerful processing capacity of this instance type, and hence shorter processing time for the same user request.

4.2 Proposed Scheduling Algorithm-CSRSGA

SaaS providers run their cloud services to profit from users using leased virtual machine resources from cloud infrastructure vendors. In this paper, we only consider the classical On-Demand Instances provision pattern such as Amazon EC2 on demand instances, which is more popular compared to Reserved Instances and Spot Instances. On-Demand provision pattern enable SaaS providers to pay for compute capacity by the time unit with no long-term commitments. In other words, SaaS providers do not have to reserve VMs in advance, and only apply for them when needed.

In this pattern, a running virtual machine instance is charged by the time it runs at a flat rate per time unit. Generally speaking, pricing is per instance-hour consumed for each instance, from the time an instance is launched until it is terminated. Each partial instance-hour consumed will be billed as a full hour [8].

In addition, we only focus on the divisible user service requests in the context of this paper, i.e., the requested load that can be continuously divided into multiple independent subtasks without precedence constraints between them. In fact, many typical cloud service related to big data sets, such as video encoding, image processing and biological sequence search (BLAST, for example), are all divisible [16,17]. Applications on platforms BOINC and distributed.net also fulfill the divisibility and independence of load grains assumptions [16].

Finally, without loss of generality, we assume the operational cost of SaaS providers only consists of the rental expense of VM instances and the penalties for violating SLA constraints.

On the basis of the above statements, we present CSRSGA (Cost-aware Service Request Scheduling based on Genetic Algorithm) to achieve optimized request scheduling by addressing the problem of leasing virtual resources, selecting an optimal subset of those resources, and mapping of user request subtasks onto selected resources. Taking into account the divisibility of user requests and the current system load, combined with the pricing model of on demand instances, CSRSGA is designed to maximize the profit by minimizing the cost of leasing resource, which depends on the number and type of initiated VM instances. Based on genetic algorithm, CSRSGA dispatches the multiple divided subtasks to the optimal VM combination in the dynamic resource pool composed of leased VM instances, and thus significantly reduces operational costs of cloud service providers without any SLA violation. It should be noted that the CSRSGA algorithm is only applicable to those divisible cloud service applications, especially those batch processing applications based on large data sets, instead of such transaction-centric cloud service as CRM or ERP.

The procedures of CSRSGA can be illustrated in Figure 3.

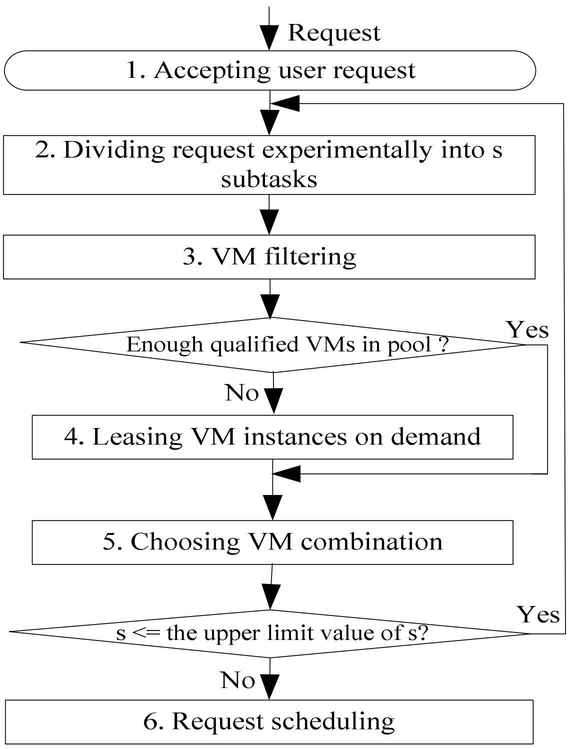

Figure 3 The procedure of CSRSGA

Next, we describe every step of CSRSGA scheduling approach in detail.

1) **Accepting user request.** As an important part of our cloud service resource provision platform, the Cloud Service Request Scheduling System is responsible for running the CSRSGA algorithm and receiving the user requests as input derived from access control model, which takes charge of user authentication and request access. In this paper, we assume that the request's execution time t_s is known [5,18].

2) **Dividing request experimentally**. CSRSGA aims to make full use of the divisibility feature of user requests and divides every request into s independent homogeneous subtasks (i.e., these subtasks have equal execution time on the same VM instance) for parallel processing, so that the unexpired idle VM instances rented from infrastructure vendors can be reused effectively.

The candidate VM instance type set used to process the user service requests is denoted by

$CVMT = \{c_1, c_2, ..., c_l\}$.

Let $s_l = \lceil t_h / deadline \rceil$, which is the minimum number of VM instances used for executing the user request so as not to violate the deadline constraint specified in the corresponding cloud service request model, and $s_u = \lceil t_l / mbdt \rceil$, which is the maximum number of VM instances used for executing the user request to obtain maximum revenue, i.e., the budget value specified in the corresponding request model, where t_h / t_l is the time required to finish the request by the VM instance with highest/lowest performance in $CVMT$, which can be figured out by formula (4) and (5).

Above all, according to the SLA constraints in the user request model, initializing the number of divided subtasks s, and let $s = s_l$. Then, repeating step 3 to step 5) until $s > s_u$ to determine the most profitable number of divided subtasks.

3) **VM filtering.** The primary goal of this step is to reduce the number of candidate VM instances used to execute subtasks to narrow down the problem search space.

Let $VRP = \{vm_1, vm_2, ..., vm_n\}$, where VRP is the resource pool consisting of unexpired VM instances leased for processing user requests from third-party infrastructure vendors, n is the ID number of the VM instance in the pool.

Traversing orderly VRP (initially empty) to find out all the VM instances meeting the following three conditions:

a) Status requirement: unexpired and idle, because only those instances satisfying this requirement are likely to accept new subtask right now.

b) Type requirement: $t_k / s < deadline$, $k = 1, 2, ..., l$, where t_k is the time required to finish the request by a c_k type VM instance. The processing time of the subtask executed by this VM instance does not violate the promised deadline constraint only when the type of candidate VM instance meets this requirement.

c) Time requirement: $rmrt > t_k / s$, where $rmrt$ is the remaining lease time of this instance. If the candidate instance cannot satisfy the time constraint, SLA violation will occur due to VM instance expiry.

Those qualified VM instances form a valid resource set denoted by $VRS = \{vm_1, vm_2, ..., vm_m\}$, where m is the number of instances in the set, and $m \le n$. If $m < s$, go to step 4), or else go to 5).

4) **Leasing VM instances.** When $m < s$, leasing new most profitable VM instances from infrastructure vendors to join the valid resource set VRS for parallel processing. Lease principle is to choose the most profitable VM instances per time unit in terms of this user request, and the lease number is $s - m$.

Let $profit\_ptu$ denotes the estimated profit per time unit obtained by processing s subtasks of the request with s VM instances of type c_k, which can be calculated as below:

$$profit\_ptu = (revenue_k - uc_k * t_k) / (t_k / s) \tag{6}$$

where uc_k is the rent cost per time unit of a type c_k VM instance, and t_k / s is the processing time of the user request which is equal to the execution time of every subtask due to the homogeneity of all subtasks of the same request. $revenue_k$ is the expected revenue, which can be figured out based on the request processing time using formula (2).

Firstly, calculating all the estimated profits per time unit of different instance types in the valid resource set which must satisfy the type requirement in 3) so that the deadline constraint of this request is not violated.

Secondly, finding out the most profitable instance type which has maximum estimated profit per unit time through comparison and leasing $s-m$ instances to join the valid resource set VRS. The lease period is $\lceil (t_k / s) / t_u \rceil$, where t_u is the pricing time unit of the instance type specified by the infrastructure vendor, usually hour.

5) **Choosing the optimal VM combination.** The optimal VM combination consists of s VM instances from the valid resource set, which enables the SaaS provider to obtain the maximum expected profit for processing the s subtasks of the user request in parallel. The expected profit $exprofit$ is the expected revenue $\exp revenue$ minus the expected cost $\exp\cos t$. This optimization problem can be expressed as the following binary integer programming problem.

$$\text{Max} \quad exprofit = exprevenue - \exp\cos t \tag{7}$$

subject to

$$ex\cos t = \sum_{i=1}^{s}\sum_{j=1}^{n} x_{ij} * uc_j * t_{ij} \tag{8}$$

$$\sum_{i=1}^{s} x_{ij} = 1, \quad j = 1,2,...,m \tag{9}$$

$$\sum_{j=1}^{n} x_{ij} = 1, \quad i = 1,2,...,s \tag{10}$$

where s is the number of subtasks derived from the same request, and m is the number of VM instances in the valid resource set. x_{ij}=1 denotes that the subtask i is executed on the instance j, 0 otherwise. uc_j is the rent cost of instance j per time unit. t_{ij} is the required execution time of subtask i run on the instance j. Supposing that c_k denotes the VM type of instance j, we can get $t_{ij} = t_k / s$, and t_k is the processing time of the request using a type c_k instance. The formula (9) ensures that a VM instance can only accept a subtask at the same time. The formula (10) ensures that a subtask can only be allocated to an instance for execution. The expected revenue $exprevenue$ can be calculated using formula (2) according to $\max\{t_{ij}\}$, because the processing time of this user request is the maximum execution time of all subtasks.

For the public cloud service operated by a SaaS provider, especially when the service becomes extremely popular all at a once, the cloud service request scheduling algorithm should respond to the high volume of user requests as soon as possible to reduce the probability of SLA violation. Therefore, genetic algorithm is introduced to accelerate the optimization problem solving process.

As a modern intelligent optimization algorithm, genetic algorithm has been widely used as an effective meta-heuristics for obtaining high quality solutions for a broad range of combinatorial optimization problems including the task scheduling problem. An important merit of genetic search is that its inherent parallelism can be exploited to further reduce its running time [19].

The solving procedure of optimal VM combination problem is shown in Figure 4. In the context of this optimization problem, the fitness function is defined as the expected profit $exprofit$, and every VM combination from the valid resource set is modeled as a chromosome (e.g., individual). The length of every chromosome is equal to m, which is the size of the valid resource set. Binary encoding is adopted, and every chromosome is a string of bits, 0 or 1. Every bit in the chromosome is a gene, which is associated with a VM instance from the valid resource set. 1 denotes that the corresponding instance is selected and 0 otherwise. As for every individual, it must conform to the sum constraint of gene value as below,

$$\sum_{i=1}^{m} gene\_value_i = s, \tag{11}$$

where $gene\_value_i$ is the bit value in the individual.

This algorithm is started with an initial population of feasible solutions randomly generated based on first-fit algorithm. Every individual in initial population must satisfy the above sum constraint, or else is considered as unfeasible solution. All of the individuals in the population are evaluated based on their fitness value, with a larger fitness value being a better mapping. Then, by applying selection, crossover and mutation operators, the best solution with maximum value of the fitness function can be found after some generations [19]. The s VM instances identified by decoding the best solution are the optimal VM combination of the current valid resource set in terms of this division scheme of this user request.

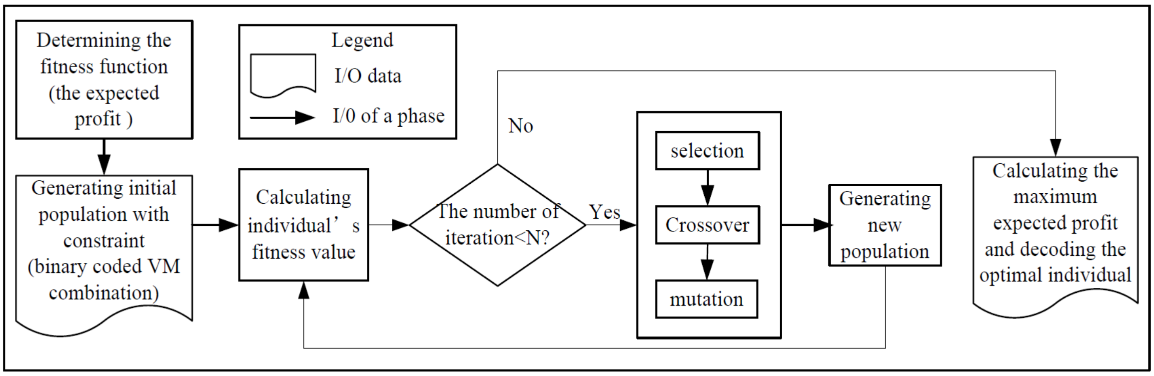

Figure 4 The optimal combination solving procedure

6) Dispatching request. Let $s_{\max}$ be the most profitable number of subtasks among all the division schemes in terms of this user request, and let the optimal VM combination composed of s instances be denoted $OVMC$, which is a subset of the valid resource set defined in the previous step. Then $s_{\max}$ and $OVMC$ can be determined by comparing the maximum expected profits of different request division schemes (i.e., different s). Finally, the user request is divided into $s_{\max}$ subtasks, and dispatched to the optimal VM combination $OVMC$ for parallel execution.

The time complexity of CSRSGA algorithm mainly consists of three parts. The complexity of VM filtering is $O(n)$, where n is the size of VRP. For the initialization of GA, the algorithm performs first-fit on a random permutation of VMs obeying the sum constraint of gene value. The complexity of initialization is $O(N*m*\log m)$, where N is the initial solution size and m is the size of VRS. The algorithm adopts classical roulette selection operator, and the time complexity of iteration operation is $O(N*G*m)$, where G is the number of generations. Therefore, the complexity of CSRSGA is $S*(O(n)+O(N*m*\log m)+O(N*G*m))$, which yields a polynomial execution time, where S is the times of request dividing.

In this section, we describe our scheduling approach for provisioning virtual resource on demand. The proposed approach exploits genetic algorithm to select the most profitable VM combination for processing user requests in reasonable time, and maximize a SaaS provider's profit by reducing the infrastructure cost and ensure that all requests are finished before their deadlines.

5. Performance Evaluation

In order to verify the effectiveness of CSRSGA algorithm proposed in this paper, we construct the following simulation experiments. Above all, we introduce the experiment setup, and then present the performance metrics for evaluation. Finally, we compare CSRSGA with three revenue-aware baseline algorithms to demonstrate the benefits of our proposed approach.

5.1 Experiment Setup

In our experiment, we model 1000 cloud service requests with different SLA constraints. Every user request is divisible and arrives in a Poisson process. The SLA parameters of different user requests are different. According to the budget constraint $budget$ in the user request model, user requests are divided into two categories: high budget class and low budget class. 20% of the requests belong to high budget class. 80% of the requests belong to low budget class. The budget values in each category follow a normal distribution. The category of next user request is random. The execution time constraint t_s specified in user request model follows an exponential distribution. The two time constraints $mbdt$ and $deadline$ are generated based on the execution time t_s. Here we let $mbdt = \alpha * t_s$, $deadline = \beta * t_s$, $\alpha < \beta$. The last constraint pr is determined jointly by the three property constraints $budget$, $mbdt$ and $deadline$.

The candidate instance set in this paper is composed of three types of Amazon EC2 On-Demand Instances, i.e., Small, Large and Extra Large (Windows Usage, California, US) [5]. The experiments are based on the capacity quantity ratio of the candidate instance types obtained by profiling and benchmarking. All the simulations are conducted on the same computer with Intel Core2 Duo CPU 2.1 GHz processor, 2.0 GB of RAM, Windows XP Professional SP3, and MATLAB7.11.0. The simulation program is written in Java based on eclipse-java-indigo-SR2-win32, and the runtime environment is JDK 1.6.0_25.

5.2 Performance metric

A SaaS provider leases VM instances from third-party infrastructure vendors to handle user requests. Under the circumstances, the type and number of initiated instances, along with their utilization rate are all associated with the operational cost and profit of the SaaS provider. Therefore, we focus on the following performance measurement metrics to evaluate our approach:

- Number of leased instances: The number of on demand VM instances leased for processing user requests. Because the rental cost of instances is calculated based on time unit, here the number of initiated instances is counted according to instance-time unit. For example, if m instances are leased for n time units, then the number of initiated instances is $m * n$.
- VM utilization rate: The total time used for processing user requests divided by the total lease time, i.e., the proportion of time that instances are busy processing requests.
- Operational profit: The net profit of the SaaS provider obtained from operating cloud services, which can be calculated using the formula $profit = revenue - cost$, where $revenue$ is the total revenue charged for processing requests, $cost$ is the total cost for renting VM instances.

- RRI: The rate of return on investment, which denotes the investment value. It can be calculated by using the formula $RRI = revenue / \cos t$.

5.3 Simulation results

We evaluate our algorithm through comparison with three baseline algorithms introduced by [5] that use homogeneous instances, RFS, RFL and RFEL, which always choose Small, Large and Extra Large on demand instances from the candidate VM instance type set to process user requests respectively. The three algorithms all guarantee that the user request be completed before the end of the property constraint $mbdt$ by leasing enough new instances to maximize the revenue of SaaS providers, and hence they can be defined as revenue-aware scheduling algorithms.

In the simulation, the population size and the number of iteration are set to be 20 and 30 respectively. The crossover rate and the mutation rate are set to be 100% and 10% respectively. The average execution time is about 1.5 second, which is mainly determined by the SLA property constraints of user request and request arrival rate. Taking into account our simulation program is far from optimal and the inherent parallelism of genetic algorithm, the execution time can be further reduced. All approaches are run for 10 times in terms of different user request data sets and all results are reported, on average.

5.3.1 Comparison on number of leased instances

We measure the number of leased VM instances of CSRSGA and compare it with the number of instances leased through three baseline algorithms in Figure 5. Our proposed approach leases fewer instances than RFS and RFL, but leases a little more than RFEL algorithm. The reason is that our algorithm takes the divisibility feature of requests into account and reuses those idle instances in the valid resource set as far as possible. Only when the current idle instances cannot meet the subtask processing requirement do we rent the most profitable instances on demand. On the contrary, the baseline algorithms always rent new instances for processing each new request ignoring resource reuse. As for RFEL, it always rent the most powerful VM instances (Extra Large) to handle user requests, and hence initiates the least instances.

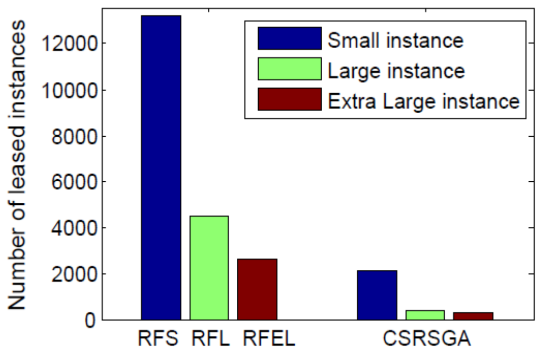

Figure 5 The number of leased instances

5.3.2 Comparison on VM utilization

As is shown in Figure 6, CSRSGA achieves higher resource utilization (approximately 80% in terms of our simulated request data set), which is mainly because that our algorithm always manages to reuse the unexpired idle VM resource for processing divided multiple subtasks in parallel. However, the three revenue-aware algorithms only focus on leasing certain type instances to maximize revenue without considering resource reuse, and thus lead to lower utilization rate (approximately 31%, 35% and 45%, on average). It should be noted that the VM utilization rate of our proposed approach is the average value of the utilization rate of three instance types.

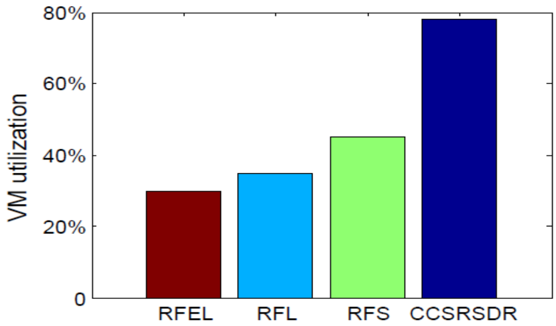

Figure 6 VM utilization

5.3.3 Comparison on operational profit

Fig 7 indicates that the proposed approach CSRSGA enables the SaaS provider to achieve more profit than other algorithms. On the one hand, the three revenue-aware algorithms intend to lease more instances to finish every request. As a result, the SaaS provider can maximize its revenue without incurring any penalty, but pay much more resource rental cost, which can be seen from Figure 5. On the other hand, due to taking no account of resource reuse, the VM instance utilization rate is significantly low compared to CSRSGA, which can be observed from Figure 6. Moreover, in the On-Demand provision pattern, the infrastructure vendor charges SaaS providers for leasing VM instances only based on the type and number of leased instances regardless of VM utilization rate. Therefore, all these factors make these revenue-aware algorithms' operational profits are lower than that of CSRSGA.

The goal of our proposed CSRSGA is also to maximize the profits of SaaS providers, but it aims to increase profit by reducing the resource rental cost instead of maximizing revenue. Taking into consideration the divisibility of user requests and the pricing model of On-Demand instances, CSRSGA makes full use of idle VM instances in the valid resource pool to achieve optimal subtasks dispatching. This may bring some penalties to SaaS providers, because of the fact that the finishing time of certain requests is greater than the *mbdt* constraint. However, the number of initiated VM instance is dramatically reduced and the VM utilization rate is significantly improved. In other words, our proposed CSRSGA balances operational revenue and resource rental cost, and hence achieves more operational profit compared with other revenue-aware algorithms.

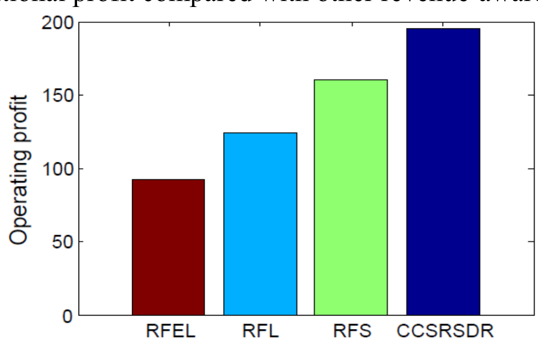

Figure 7 Operational profit

5.3.4 Comparison on RRI

From Figure 8, we can see that CSRSGA outperforms greatly the alternative algorithms in RRI, which was mainly derived from the cost savings resulting from high VM utilization. Therefore, our proposed scheduling approach is more attractive to SaaS providers.

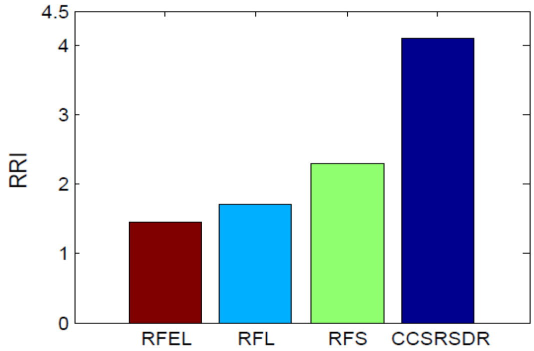

Figure 8 RRI

To sum up, the simulation experiment results show that CSRSGA provides a more cost-effective solution for user service request scheduling, and hence verify its effectiveness.

6. Conclusion

SLA is the focus of users' attention, and earning profit is the principal driving force for SaaS providers. In order to satisfy the benefits of both SaaS providers and end users, it is of importance to design a cost-effective user service request scheduling algorithm in cloud computing scenario. To deal with this problem, we first establish a user request model under SLA constraints, and then present a cost-aware service request scheduling approach CSRSGA, which takes the divisibility feature of user requests and dynamic resource reuse into consideration. It can identify the most profitable VM combination of the valid resource set using genetic algorithm to achieve optimal subtask dispatching in reasonable time, and thus maximizes the operational profits of SaaS providers without violating any SLA constraint. The experiment results indicate that our proposed CSRSGA is superior to the alternative revenue-aware algorithms and provides a cost-effective solution for cloud service request scheduling.

In building on the research undertaken in this paper in the future, we will investigate the cost-aware service request scheduling problem taking into account user satisfaction and SLA negotiation process in cloud computing environment. In addition, we plan to consider other pricing strategies such as Amazon spot pricing for maximizing a SaaS provider's profit.

Acknowledgements

The work is supported by the NSFC (61272521), NSFC (61202435), SRFDP (20110005130001), FIRGNNSFC (61121061), NCET (100263), BMNSF (4132048) and CPSF (2011M500226).

References

[1] Armbrust, M., et al. (2010) A view of cloud computing. Communications of the ACM, **53,** 50-58.

[2] NIST Special Publication 800-145. (2011) A NIST definition of cloud computing. National Institute of Standards and Technology, Gaithersburg, MD 20899-8930, USA.

[3] Hajjat,M., Sun, X., Sung, Yu-Wei. E., Maltz, D. and Rao S. (2010) Cloudward Bound: Planning for Beneficial Migration of Enterprise Applications to the Cloud. Proceedings of SIGCOMM 2010, New Delhi, India, Aug 30-Sep 3, pp. ~243-254. ACM, NewYork.

[4] S.C. Yu, C.,Wang, K. Ren, and W.J., Lou. (2010) Achieving Secure, Scalable, and Fine-grained Data Access Control in Cloud Computing. Proceedings of INFOCOM 2010, San Diego, CA, USA, 15-19 March, pp.~534-542. IEEE, Piscataway, NJ.

[5] J.C., Chen, C., Wang., B.B., Zhou, L. Sun, Y. C., Lee and A. Y. Zomaya. (2011) Tradeoffs between Profit and Customer Satisfaction for Service Provisioning in the Cloud. Proceedings of HPDC 2011, San Jose, California, USA, 8-11 June, pp.~229-238. ACM, NewYork.

[6] Y.C., Lee, C., Wang, A. Y., Zomaya and B. B., Zhou. (2012) Profit-driven scheduling for cloud services with data access awareness. J. Parallel Distrib. Comput. **72,** 591-602.

[7] NIST Special Publication 500-292. (2011) NIST Cloud Computing Reference Architecture. National Institute of Standards and Technology, Gaithersburg, MD 20899-8930, USA.

[8] M., Mao, Humphrey, M. (2011) Auto-Scaling to Minimize Cost and Meet Application Deadlines in Cloud Workflows. Proceedings of SC 2011, Seattle, Washington, USA, 12-18 November, pp.~1-12. ACM, NewYork.

[9] Sherwani1, J., Ali, N., Lotia1 N., Hayat, Z. and Buyya, R.(2004) Libra: a computational economy-based job scheduling system for clusters. Softw. Pract. Exper, **34,** 573-590.

[10] Vanderster, D.C., Dimopoulos, N.J., Rafael P.H. and Sobie, R.J. (2009) Resource allocation on computational grids using a utility model and the knapsack problem. Future Generation Computer Systems, **25,** 35-50.

[11] Popovici, F.I. and Wilkes, J. (2005) Profitable services in an uncertain world. Proceedings of SC 2005, Seattle, Washington, USA, 12-18 November, pp.~36. IEEE, Piscataway, NJ.

[12] Bonvin, N., Papaioannou, T.G. and Aberer, K. (2011) Autonomic SLA-driven Provisioning for Cloud Applications. Proceedings of CCGrid'2011, Newport Beach, CA, USA, 23-26 March, pp.~434-442. IEEE, Piscataway, NJ.

[13] Wu, L.L., Garg, S.K., Buyya, R. (2011) SLA-based Resource Allocation for Software as a Service Provider (SaaS) in Cloud Computing Environment. Proceedings of CCGrid 2011, Newport Beach, CA, USA, 23-26 March, pp.~195-204. IEEE, Piscataway, NJ.

[14] AuYoung, A., Grit, L., Wiener, J. and Wilkes, J. (2006) Service contracts and aggregate utility functions. Proceedings of HPDC 2006, Paris, France, 19-23 June, pp.~119-131. IEEE, Piscataway, NJ.

[15] Zhu, Q. and Agrawal, G. (2010) Resource Provisioning with Budget Constraints for Adaptive Applications in Cloud Environments. Proceedings of HPDC 2010, Chicago, Illinois, USA, 20-25 June, pp.~304-307. ACM, NewYork.

[16] J. Berlińska, M. Drozdowski. (2011) Scheduling divisible MapReduce computations. J. Parallel Distrib. Comput., **71**, 450-459.

[17] Sangho Yi, Artur Andrzejak, and Derrick Kondo. Monetary Cost-Aware Checkpointing and Migration on Amazon Cloud Spot Instances. IEEE Transactions on Services Computing, **5,** 512-524.

[18] Garg, S.K., Yeo, C.S., Anandasivam, A, Buyya, R. (2011) Environment-conscious scheduling of HPC application on distributed cloud-oriented data centers. J. Parallel Distrib. Comput., **71,** 732-749.

[19] Omara, F.A., and Arafa, M.M. (2010) Genetic algorithms for task scheduling problem. J. Parallel Distrib. Comput., **70,** 13-22.