Hindawi Publishing Corporation
Mathematical Problems in Engineering
Volume 2015, Article ID 747392, 9 pages
http://dx.doi.org/10.1155/2015/747392

Research Article

A Revenue Maximization Approach for Provisioning Services in Clouds

Li Pan[1] and Datao Wang[2]

[1]*School of Computer Science and Technology, Shandong University, Jinan 250101, China*
[2]*Jinan Resident Office of CNAO, Jinan 250011, China*

Correspondence should be addressed to Li Pan; panli@sdu.edu.cn

Received 27 March 2015; Accepted 16 June 2015

Academic Editor: Bao Rong Chang

With the increased reliability, security, and reduced cost of cloud services, more and more users are attracted to having their jobs and applications outsourced into IAAS data centers. For a cloud provider, deciding how to provision services to clients is far from trivial. The objective of this decision is maximizing the provider's revenue, while fulfilling its IAAS resource constraints. The above problem is defined as IAAS cloud provider revenue maximization (ICPRM) problem in this paper. We formulate a service provision approach to help a cloud provider to determine which combination of clients to admit and in what Quality-of-Service (QoS) levels and to maximize provider's revenue given its available resources. We show that the overall problem is a nondeterministic polynomial- (NP-) hard one and develop metaheuristic solutions based on the genetic algorithm to achieve revenue maximization. The experimental simulations and numerical results show that the proposed approach is both effective and efficient in solving ICPRM problems.

1. Introduction

Cloud computing is holding a promising approach to deliver on-demand computing services to a wide range of consumers in a pay-as-you-go way in these years. With virtualization as a key enabler, cloud computing delivers Infrastructure As A Service (IAAS) that integrates computation, storage, and networking resources in a virtualized environment [1]. Virtualization technology makes the independence of applications and servers feasible through consolidating multiple applications and jobs running in different virtual machines (VMs) on a single physical machine (PM).

Virtualization not only brings cloud providers efficiency gains in terms of processing power but also saves electric power, space, and cooling, since the number of physical machines running is greatly reduced [2]. Recently, there is a significant increase in both the supply and demand sides of this new market for IAAS cloud services. It represents a new business model where clients buy job execution services from clouds while the cloud providers gain revenues. Such IAAS environments offer elastic job execution services with flexible QoS in a way that they can be measured, thus allowing for a pricing mechanism to be enforced. Clients often have different preferences on service qualities and each client is characterized by a willingness-to-pay function. This function is usually parameterized with both the clients own payment willingness and QoS of the offered service. Based on payments from clients, cloud providers try to maximize their financial revenue through provisioning services.

The above problem is defined as IAAS cloud provider revenue maximization (ICPRM) problem in this paper. We address this problem through provisioning services of flexible QoS levels to clients. Briefly speaking, we equip the provider with an admission control mechanism that helps the provider to determine which subset of clients to admit and what QoS levels to offer to each admitted client, in order to maximize his revenue. We formulate the revenue maximization problem here as a relaxed Multiple-Choice Knapsack Problem (MCKP), which is a variant of the classical 0/1 Knapsack Problem (KP). Since to find an exact solution for a MCKP is an NP-hard problem, in this paper we propose a genetic algorithm to obtain a feasible near-optimal solution. And we evaluate our revenue maximization mechanisms for provisioning services in a simulated environment to illustrate the efficiency and effectiveness of the proposed approach.

The remainder of the paper is organized as follows. Section 2 gives an overview of the IAAS cloud computing environment. The optimization problem and the proposed genetic optimization algorithm are presented in Sections 3 and 4. Section 5 outlines simulation experimental results. Related work is discussed in Section 6 while concluding remarks are found in Section 7.

2. System Model

This section presents an overview of the cloud platform environment and the client model assumed in our work, which is based on an IAAS model. For clarity, in this paper we focus on a simple type of IAAS service: provisioning virtual machine in a provider's cloud datacenters to run jobs, especially batch type jobs submitted by clients.

2.1. IAAS Cloud Platform Description. We consider a scenario where a cloud provider hosts multiple clients' jobs in a shared IAAS platform. Clients who have jobs to run gain access to the cloud resources by requesting services from the provider. The service proposed by the cloud provider in our approach is offering infrastructures to the clients in order to run their jobs. Whether jobs are run in virtual machines or raw physical machines, they are transparent to clients and they only care for the quality of the service the cloud provider offered to run their jobs. And the role of this work is to help the provider to maximize its revenue while proposing and provisioning services.

A high-level structural view of the IAAS environment considered in this work is depicted in Figure 1. As shown in the figure, an IAAS provider's physical infrastructure consists of a number of computing servers, which are equipped with physical resources such as CPU, memory, and I/O bandwidth. On top of the physical infrastructure is the virtualization layer created by isolated VMs. Virtualization mechanisms partition the physical resources into multiple isolated virtual machines. Specifically, in this paper for clarity we only consider the number of CPUs allocated to a virtual machine for provisioning services to run jobs, as the processing capacity measurement criteria. Thus a VM's processing performance is characterized by its allocated virtual CPU cores. The different numbers of CPUs allocated to a client's VM would result in different service qualities for running jobs. It is the responsibility of the *service provision manager* to admit jobs submitted by clients and determine how many CPUs are required to provision virtual machines for running the submitted jobs.

The above-described hosting environment in fact forms a cloud service market where a cloud provider sells IAAS services rather than raw machine resources and clients make payment as returns to the provider for running their jobs. Each client desires a performance bound, that is, QoS, for its job execution service, and pays corresponding prices. In this work we consider a bidding approach, upon which each client submits a bid to the cloud provider to obtain job execution services. Each client has a utility function that gives its budget value as a function of a range of service quality levels. We will discuss the utility functions in the following subsection. The cloud provider selects clients for provisioning capacities based on clients' bids for submitted job execution service requests. In the cloud provider's side, a service provision manager is responsible for admitting clients to run their jobs. When faced with a bunch of clients, the admission control includes deciding which clients to admit, in what price to provision the services, and in what QoS levels. We believe that for a batch type job execution cloud service one performance measure that matters most to the clients is *job response time*, that is, expected job completion time [2]. Thus in this paper we will take the job response time as the criteria of service QoS levels.

Given the framework described, we define the provider's business objective, formulated as ICPRM problem, as the optimal provisioning of capacities to VMs to host clients' jobs so as to maximize his overall revenue.

2.2. Utility Functions. In this subsection we discuss the utility functions, based on which the clients specify bids and the provider makes admission control and virtual machine provision decisions.

In this work we assume that a client has a flexible but constrained requirement on the response time of his jobs. This means that a client has a rigid deadline requirement before which the job must be finished; otherwise he would not pay for the job execution service. On the other side, the client would pay more for a job finished earlier than its rigid deadline. For example, a client may have 20 hours as the deadline for a job he submits to a cloud platform, but he would pay more if the job will be finished sooner. Thus, for each client, a utility function specifies the value he is willing to pay to the provider for a range of service quality levels—expected job completion time acceptable to the client. This utility function is also called the client's willingness-to-pay function.

The formulation of utility functions must be simple, rich, and tractable. We assume that a client has a basic utility gain (or revenue) from finishing the job and a convexly increasing utility loss for the delay in service response time t_i. And we model utility function $u_i(t_i)$ of a client for a job execution service provided as

$$u_i(t_i) = a_i + b_i * \left(1 - \left(\frac{t_i - t_{i,\min}}{t_{i,\max} - t_{i,\min}}\right)^{\alpha_i}\right), \quad (1)$$

where a_i, b_i, α_i, $t_{i,\min}$, and $t_{i,\max}$ are user-defined variables, with a_i denoting the base payment for running the job, b_i denoting the floating return for accomplishing the job earlier than the deadline, $t_{i,\min}$ denoting the earliest job finish time reasonable, and $t_{i,\max}$ denoting the maximum completion time acceptable by the client i, beyond which he would not make payment for the service. And α_i is a client specific parameter characterizing how he is sensitive to service response time delay. A set of example utility functions is shown in Figure 2. By modeling in this way, a client's utility functions u_i are concave and monotonically decreasing in t_i, which is reasonable in batch type job execution domains.

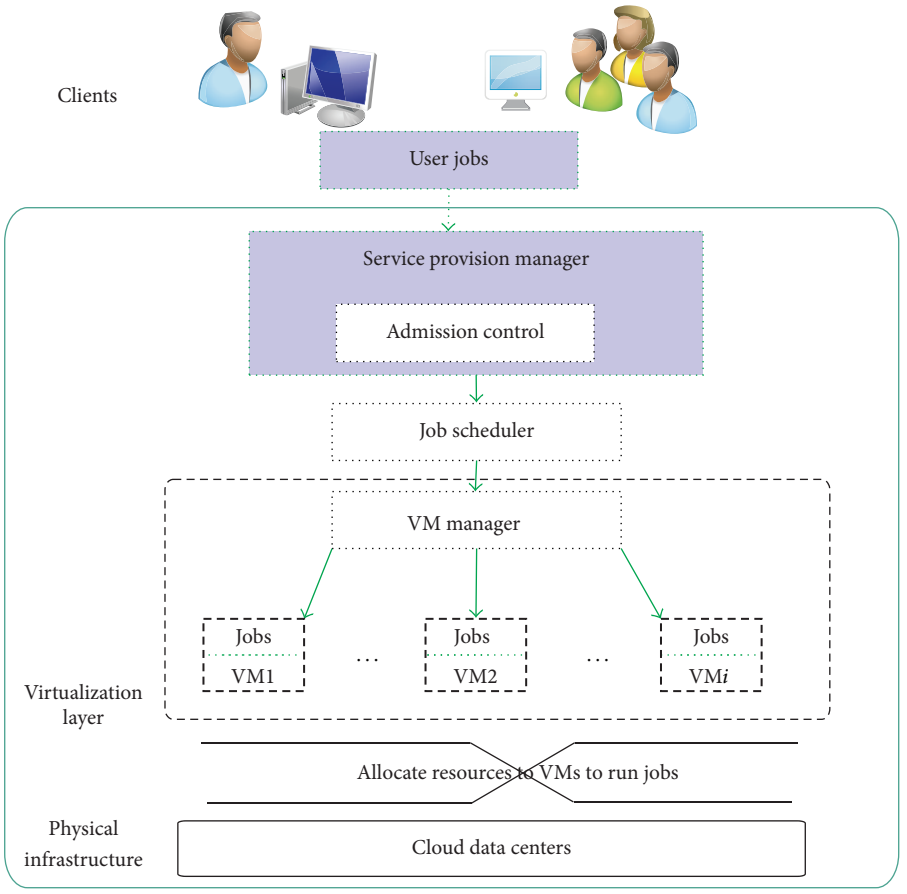


Figure 1: System model.

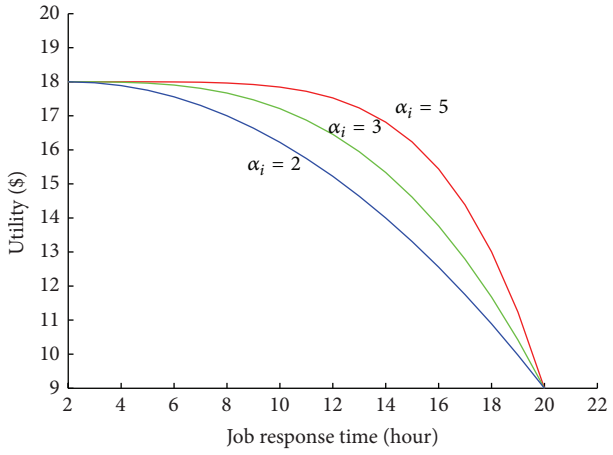


Figure 2: Example utility functions. α_i is set by a client and denotes how he is sensitive to service response time delay.

2.3. How Much to Provision: Modeling Batch Type Jobs Execution in Clouds. The goal of our provision mechanism is for the IAAS provider to optimally decide the QoS levels and allocate specific number of CPU cores to provision virtual machine to run clients' jobs. A significant problem for the provider to provision CPUs to run a job is to decide how many CPU capacities are needed to meet a specific QoS demand, that is, job response time in this paper. This is a reverse problem of determining the completion time of a job given the physical capacity of the machine on which it runs. Estimating job completion time given specific processing capacity is a complex problem that has been extensively researched and it falls beyond the scope of this paper. We acknowledge that it is sometimes difficult to estimate accurate execution time given some types of jobs. And whether our revenue maximization driven virtual machine provisioning approach is effective or not is highly dependent on the accuracy of the estimation of job execution time. However, for batch type jobs we discussed in this paper, we argue that it is possible to build fairly accurate models. This model can be built by benchmark methods. And in this paper, for clarity reasons, we assume that job execution time T has a certain relationship with the number of CPUs allocated to the VM it runs, represented as n, as follows:

$$T(n) = \frac{t}{\ln(n) + 1}, \tag{2}$$

where t corresponds to the time it would take to complete the job if only one general CPU core is provisioned to a VM to run the job. With this equation we model the fact that job execution time will reach a saturation point and will improve only marginally after reaching a certain number of CPUs.

3. Revenue Maximization Problem Formulation

3.1. Problem Formulation. We model our IAAS service platform to have a certain number of available identical CPU cores, denoted as C, which are running in a cluster of machines maintained by the cloud provider. Consider a given arbitrary set of clients I such that the ith client ($i \in I$) has a job to execute, the acceptable response time range $[t_{i,\min}, t_{i,\max}]$, and the utility function u_i for specifying payment based on the guaranteed service response time t_i, that is, job completion time. It is assumed that the variable values of response time are not taken continuously from the acceptable range of a client but discretely. This assumption is reasonable due to the fact that end users are usually not sensitive to response time difference of a too small time scale; that is, end users may feel no difference between spending, for example, 1 minute and spending 2 minutes to get a job finished [2]. For example we may consider a granularity of 30 minutes.

As described above, our objective is to maximize the provider's revenue within its CPU capacity constraints, and the mathematical formulation of the ICPRM is as follows:

$$\begin{aligned} \text{maximize} \quad & \sum_{i \in I} x_i u_i\left(t_i\right), \\ \text{subject to} \quad & t_i \in T_i = \left\{\tau_{i,1}, \ldots, \tau_{i,e}\right\} \subset \left[t_{i,\min}, t_{i,\max}\right], \\ & \qquad\qquad i \in I, \quad (3) \\ & x_i \in \{0,1\}, \quad i \in I, \\ & \sum_{i \in I} T^{-}\left(t_i\right) \le C, \quad i \in I, \end{aligned}$$

where $T^{-}(t_i)$ denotes CPU provision decision function for the ith client, x_i is a pseudo-Boolean integer decision variable to determine whether the ith client is admitted or not, and t_i is a variable to denote the response time for the ith client's job. The CPU provision decision function $T^{-}(t_i)$ is a reverse function of (2). The job response time set $T_i = \{\tau_{i,1}, \ldots, \tau_{i,e}\}$ is derived discretely from the ith client's acceptable range $[t_{i,\min}, t_{i,\max}]$. We may without loss of generality assume $\tau_{i,1} = t_{i,\min}$ and $\tau_{i,e} = t_{i,\max}$.

The above ICPRM problem is a relaxed Multiple-Choice Knapsack Problem (MCKP), where there are groups of items with the constraint that exactly an item can be picked from each group, and for ICPRM problem the restriction of picking exactly one item from each group is relaxed; that is, you can either pick one item from a group or leave it. Leaving a job unserviced models the situation that the provider denies a client either because servicing it is nonprofitable or because there are not enough CPU capacities left.

3.2. Optimization Technique. Now let us study the above profit maximization problem and discuss the optimization techniques for solving it.

Theorem 1. *The optimization problem ICPRM is NP-complete.*

The theorem's proof is a straightforward reduction from the 0-1 Knapsack Problem. Since we only consider a finite number of QoS level alternatives, it is tempting to do a complete enumeration to determine the optimal solution for a small problem size. However, with the increasing number of potential clients faced by the provider, finding exact solutions for this NP-complete problem would incur huge and usually unacceptable computation cost. Thus, we resort to heuristic algorithms for near-optimal solutions. There exist a number of heuristics in the literature for MCKP and KPs in general. For example, greedy approaches have been proposed to find near-optimal solutions of KPs [3, 4]. Even though greedy approaches are simple to implement and are widely used, the optimality cannot often be guaranteed when trapped in local optimum. Thus, in this work we propose a metaheuristic genetic algorithm (GA) instead of greedy heuristics to handle this computationally costly optimization problem.

4. Genetic Algorithm Optimization for ICPRM

In the previous section we have shown that the optimization problem ICPRM is NP-hard. We now develop a metaheuristic genetic algorithm in order to find a near-optimal solution to this revenue maximization problem.

4.1. Review of Genetic Algorithm (GA). Genetic algorithms (GA) have been considered as effective techniques for solving complex operational research problems. A genetic optimization algorithm can be considered as an iterative process to search the best solutions from a predefined sized population of chromosomes for a given problem. The search usually starts from an initial randomly generated population of individuals. During every evolution step, a fitness function is used to evaluate individuals, and then the operators of reproduction, crossover, and mutation are invoked to create offsprings. In these steps the individuals with high fitness value will have a higher probability to reproduce.

4.2. GA for ICPRM Problem. In this section, we describe the formulation of a genetic optimization algorithm for ICPRM problem. To cope with the ICPRM problem, we propose the GA features relevant to solution encoding, fitness value, elitist reproduction, and crossover operation during the evolution process.

4.2.1. Solution Encoding. When applying genetic optimization algorithms to solve real-life operational application problems, the hardest step is usually how to encode the solutions as chromosomes. The chromosomes are often represented by strings and thus genetic operators can be applied on them. In this work, we propose an encoding scheme for revenue maximization as shown in Figure 3.

A solution is represented by a string of length equal to the number of variables, which is the number of clients currently requesting services from the provider. Each position, i, of the string, which represents the ith gene in the chromosome, can take any integer from $\{0, \ldots, e_i\}$, where an integer "0" in position i represents the ith client that will not be admitted

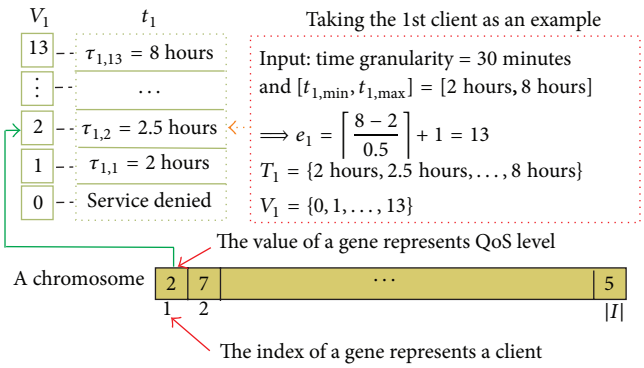


FIGURE 3: Solution encoding.

while one of the other integers in the same position, say j, represents the ith client that will be admitted and serviced with job execution response time $T_i[j]$, which is the jth element in vector T_i, that is, $\tau_{i,j}$. This is illustrated by the following example.

Example 2. Let $|I| = 5$ and $e_i = 5$ for all i. Then one string representing a chromosome is an element of $\times_{i=1}^{5}\{0, 1, 2, 3, 4, 5\}$. A particular instance $(5, 2, 4, 0, 1)$ represents the solution which admits the 1st client's job with completion time guaranteed at $t_1 = \tau_{1,5}$, the 2nd with $t_2 = \tau_{2,2}$, the 3rd with $t_3 = \tau_{3,4}$, and the 5th with $t_5 = \tau_{5,1}$, respectively, while denying the 4th client's service request.

4.2.2. Fitness and Constraints. The fitness function measures to what extent the value of an individual chromosome satisfies the objective. For an individual feasible solution, its fitness value is the overall revenue gained by admitting the set of clients with the QoS levels denoted in it. Besides fitness value, a feasible solution must meet CPU constraints in our ICPRM problem. This means that during the whole evolutionary process all the constraints should be met anytime. This requires the operations of individual chromosome initialization, crossover, and mutation in our approach be modified accordingly. Thus, while creating the initial population, if the required overall amount of CPU capacities of a randomly generated individual violates the constraints, it must be rejected and regenerated. Similarly, during the crossover and mutation process, if an offspring or mutated individual violates the CPU constraints, it should be also given up and generated again.

4.2.3. Elitist Reproduction. During the evolution process, our GA identifies itself with its elitist reproduction schemes, which we will discuss below. At the beginning of GA, the initial population P^0 is created by randomly selecting N_p members from the candidate set, where N_p is the size of the population. When creating mating pools MP, in our approach an *elitist reproduction* replaces the usual probabilistic reproduction, by firstly cloning N_r top solutions. N_r is usually fixed to 10% of the population size. The advantage of using an elitist reproduction is that it can help prevent the loss of good solutions once they are found and thus the best solution is monotonically improving from generation to generation. After elitist reproduction, *tournament selection* is then invoked, during which a group of k members are selected to create a tournament. Then, the highest fitness valued individual from the tournament is picked out and placed in the mating pool. This process continues until the mating pool reaches the predefined size.

4.2.4. Evolution Process. During every evolution process, we apply standard cloning, crossover, and mutation operations over the individuals in the mating pool to create offsprings. While doing cloning operation, an individual is directly copied from the mating pool to the offspring population. After cloning operations, we use crossover operations to exchange the randomly selected fields (QoS level indexes) of two randomly selected parents to obtain two offsprings. Concerning mutation, we randomly replace QoS level indexes encoded on a chromosome by other QoS level indexes from the permissible QoS domains of a client. And the operations of cloning, crossover, and mutation continue until the offspring population with size N is created. The exit criteria in our genetic algorithm can be defined by a maximum number of generations (evolutions) or the situation occurs when the best feasible solution has not been updated for a few generations.

The genetic algorithm described above is summarized by the pseudocode in Algorithm 1. Note that the best feasible solution is updated at each generation.

5. Performance Analysis and Discussion

In order to evaluate the effectiveness and the efficiency of the proposed genetic based revenue maximization algorithm for the ICPRM problem, a simulation framework is implemented. We have changed the size of submitted jobs and the number of alternative job response time levels to test the performance of the proposed approach. Besides, we compare our proposed genetic optimization based approach with other methods. The detailed simulation setups, baseline heuristics, and numerical results of this implementation are explained next.

5.1. Experimental Methodology. For simulations, experimental settings and model parameters are chosen based on true-to-life IAAS cloud environments. Since higher load factors (the total demanded capacity divided by total capacity) increase the importance of an optimal capacity provision approach, for each test case the number of total available capacities is randomly set in a way that the load factors vary between 1.1 and 1.5. The number of clients varies between 10 and 100. For simplicity of implementation and demonstration, we assume that each client has the same range of acceptable job response time which is from 2 hours to 8 hours. Thus, different time granularity would result in varied number of alternative job response time levels. For example, when taking 30 minutes as the granularity, the number of alternative response time levels would be 12; but when taking 1 hour as the granularity, the number of alternative response

LET
N_p be the size of the GA population;
P_c be the cross probability and P_m be the mutation probability;
N_r be the elitist reproduction percentage;
INITIALIZE
set best revenue value so far to zero;
create the initial population P^0 by randomly selecting N_p feasible individuals from $V_p = V_1 \times V_2 \cdots \times V_{|I|}$;
evaluate the fitness value of each individual in P^0 and sort them by decreasing order;
update best solution so far as the best individual in P^0 and set best revenue value correspondingly;
WHILE NOT(Reached the exit criteria)
  P^{i+1} = ElitistReproduction(P^i, N_r); //to include the first N_r solutions in P^{i+1};
  MP = TournamentSelection(P^i);
  DO UNTIL Size(P^{i+1}) = N_p
    Q_c = Crossover(MP, P_c);
    Q_m = Mutation(MP, P_m);
    $P^{i+1} = P^{i+1} \bigcup Q_c \bigcup Q_m$;
  END UNTIL
  UPDATE best solution and best revenue value so far;
END WHILE
OUTPUT best solution and best revenue value;

Algorithm 1: Genetic algorithm for ICPRM problem.

time levels would be 6. And in the following experiments, we change the granularity to get varying number of response time levels.

5.1.1. Heuristics for Comparison. In order to study the runtime performance of our genetic algorithm for revenue maximization, we implement it along with a greedy approach. The greedy approach is based on a linear search of all feasible QoS levels and picks the highest-valued QoS level for each client. The value of each QoS level is calculated as the ratio between the revenue gained from it and the number of CPUs needed. And then the set of clients are sorted based on their highest values of feasible QoS levels. Finally the subset of clients with their guaranteed job response time levels are selected by packing them into the limited CPU capacities one by one from the sorted set of clients (from the highest to the lowest), until no more clients can be admitted. In this paper, we call this heuristic Greedy-HEU.

We implement all the algorithms in C programming language. The parameters in genetic based algorithm are set as follows:

(i) The size of the population is 100.

(ii) The maximum generation is 100.

(iii) The cross probability is 0.7 and the mutation probability is 0.1.

To avoid biasing results due to randomness, our GA executions are repeated 10 times, and average values are used.

5.2. Efficiency. We take the average CPU execution time to test the efficiency of the proposed genetic optimization based approach for the revenue maximization problem. We consider that the size of the jobs submitted by clients varies from 10 to 100 with increment of 10, and the number of alternative job response time levels changed from 6 to 14, with increment of 2. We ran all the simulations on an Intel 1.70 GHz PC with 3.69 G of RAM under Windows.

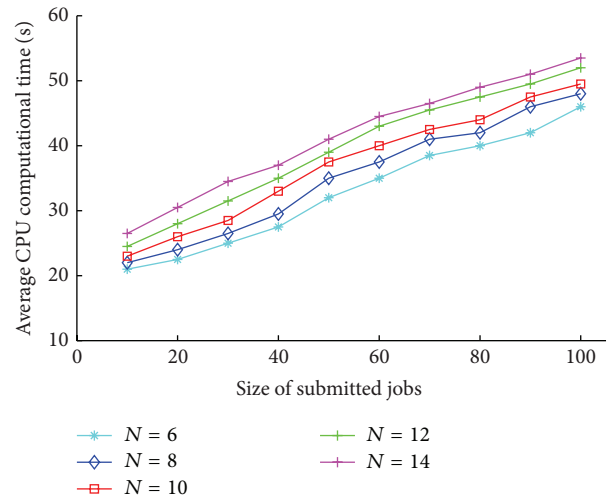


Figure 4: Average CPU computational time of our GA revenue maximization algorithm for different problem sizes. The parameter N is the size of the alternative response time levels.

We first test how our proposed genetic algorithm works for different problem instance sizes. The summary of results is shown in Figure 4. It can be seen from the figure that with the growth of job population size the computation time under the same number of alternative job response time levels increases. However, the computation time increases very slowly. Similarly, with the increment of alternative job execution time levels, the computation time under the same

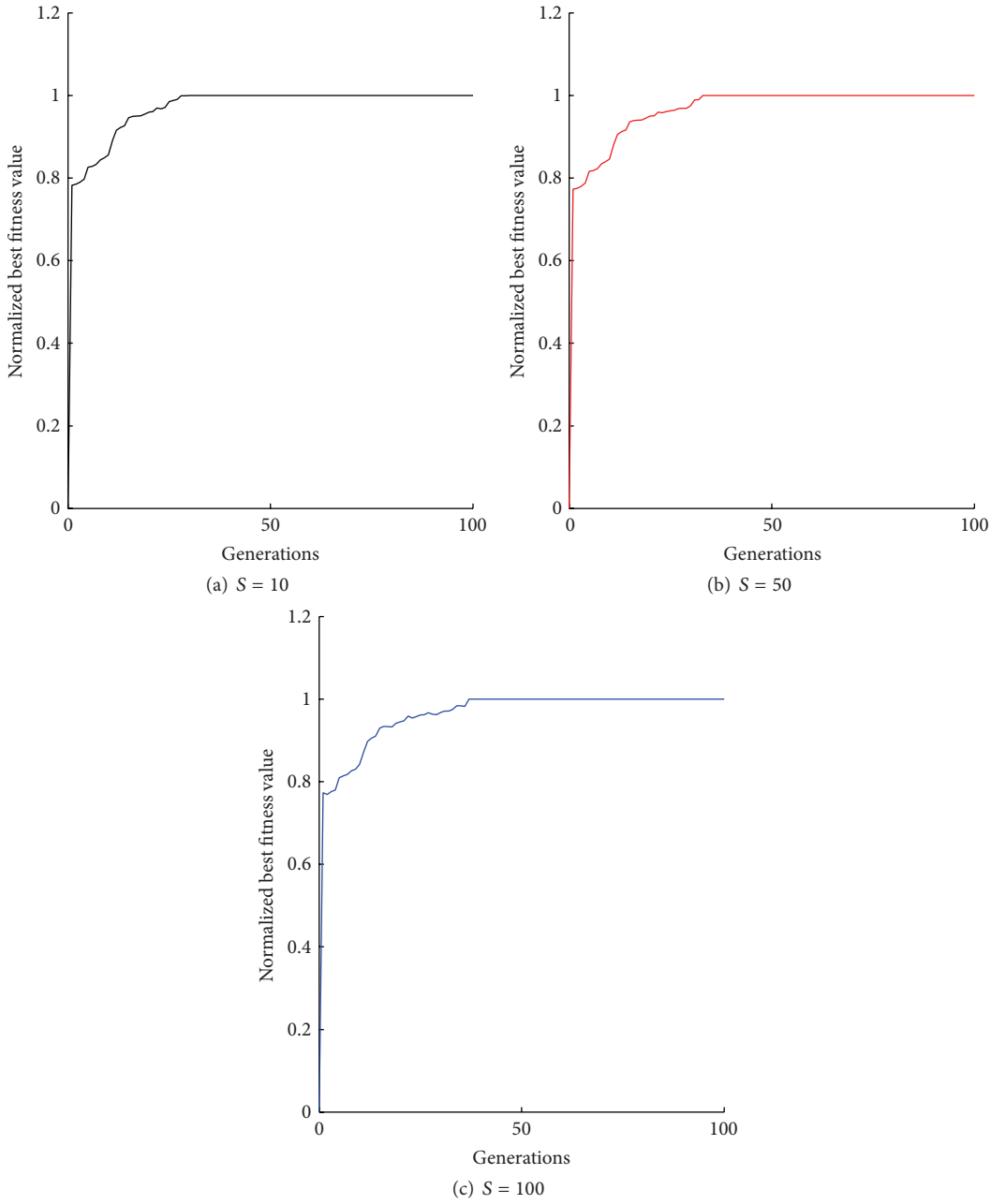
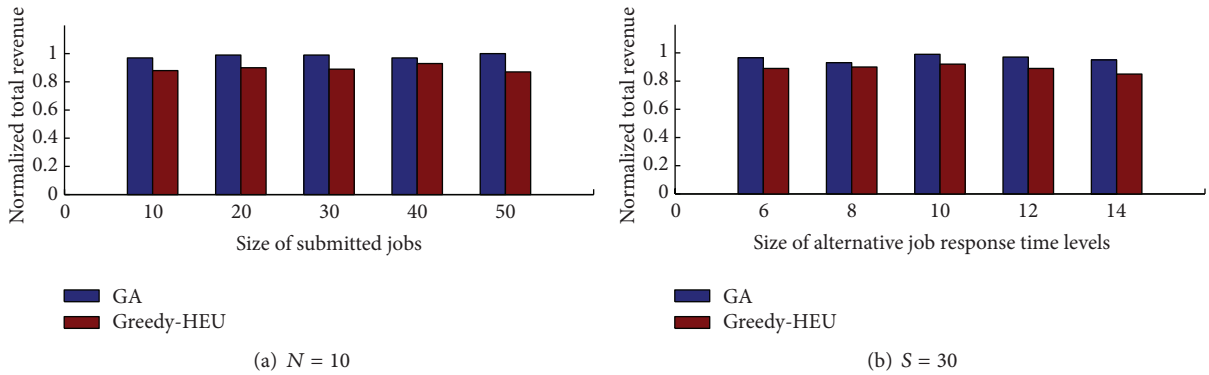


Figure 5: Examples of evolution trace. The parameter S is the size of the submitted jobs.

number of job population sizes also increases slowly. And for the biggest problem size tested in our simulation framework, the execution time of our proposed algorithm is no bigger than 60 seconds. This solving scale suits the requirements of most cloud provider revenue maximization problems.

Besides, the numbers of generations for our proposed genetic algorithm to obtain near-optimal solutions are less than 30 for the different problem instance sizes tested. Figure 5 shows three examples of the execution trace of our GA for solving ICPRM problems. The three examples are all parameterized as the alternative response time levels are 10 but with different job number sizes, as shown in the figure. As can be seen, it converges to the near-optimal solution, after about 30–40 generations in these examples. Overall, the proposed GA is efficient since near-optimal solutions for problems of reasonable sizes are obtained in at most 60 seconds and hence the approach is suitable for online implementations.

5.3. Effectiveness. We also conduct experiments to test the effectiveness of the proposed GA based optimization approach to find the global optimal solution for ICPRM problem. Figure 6 reports the optimal result obtained by our proposed GA optimization algorithm, as well as Greedy-HEU



(a) $N = 10$ (b) $S = 30$

FIGURE 6: Normalized revenue of the provider for different algorithms. The parameter N is the size of the alternative response time levels and S is the size of the submitted jobs.

method described earlier. The *normalized* total revenue is calculated as the percentile between the obtained results and the "exact optimal global result," which can be obtained by exhaustive search computing. Each result value is an average of 10 test cases. The detailed settings of our experiments are as follows.

First, the numbers of alternative job response time levels are fixed at 10, and the job population size varies from 10 to 50 with an increment of 10. The corresponding experimental results are shown in Figure 6(a). It can be seen that the GA optimization algorithm can find near-optimal solutions compared with "exact optimal global result," while Greedy-HEU cannot produce near-optimal results at most times. The result value of our GA optimization algorithm ranges between 90% and 100% when the job population size varies from 10 to 50, while the numbers of job response time levels are 10. It indicates that the size of the job population does not affect the quality of the proposed genetic optimization approach in obtaining the near-optimal solutions for the ICPRM problem.

Second, the job population size is fixed at 30, and the numbers of job execution time levels vary from 6 to 14 with an increment of 2. The results of this condition are shown in Figure 6(b). It can be seen that our GA can also find near-optimal solutions. The result value of our GA optimization algorithm ranges between 90% and 100% when the job response time levels vary from 6 to 14, while the job population size is 30. It indicates that the scale of job execution time levels does not affect the quality of the proposed GA in obtaining the near-optimal solutions for the revenue maximization problem.

5.4. Experimental Results and Comparison Discussion. For the ICPRM problem, Figure 6 demonstrates the normalized total revenue of the provider using our proposed GA and Greedy-HEU algorithms. It can be seen that the proposed approach can find near-optimal solutions, while Greedy-HEU cannot produce near-optimal results at most times. Though exhaustive search methods can help find exact global optimal solutions to the ICPRM problem, the CPU computational time needed by those methods becomes intractable and unacceptable with the growth of the problem instance size. For example, it takes about 20 minutes to get the global optimal solution for the ICPRM problem with the job population size being 10 and the number of job response time levels being 10. But when we increase the number of response time levels to 12, the exhaustive search method needs about 2 hours to get the optimal solution. Thus, the exhaustive search methods are not practicable and applicable to the cloud revenue maximization problem. Figures 4 and 5 report that for different problem instance sizes our proposed GA approach converges quickly and can find near-optimal solutions within 60 seconds. Above all, we can conclude that our proposed GA is both efficient and effective in finding near-optimal solutions for the ICPRM revenue maximization problems.

6. Related Work

The problem of revenue maximization in the cloud computing filed has become a crucial issue and attracted significant attention in the last few years. Below we provide a review of most relevant prior work.

Jin et al. propose a dynamic game to achieve Nash equilibrium in a distributed manner and tackle cloud price competition problem with Bertrand game modeling to maximize cloud provider's profit [5]. Their approach can help cloud providers compete for profit maximization in an oligopoly market. The work in [6] studies a cloud computing market where a cloud provider rents a set of computing resources from Windows Azure operated by Microsoft, and it proposes a stochastic programming model with two-stage recourse. The work in [4] aims at maximizing the revenues from SLA of multiclass systems where each class of request is assigned to a number of dedicated homogeneous servers to provide performance guarantees. The number of servers is evaluated by modeling each physical server as a G/G/1 queue, although the solution is determined by a very simple greedy approach.

Tsakalozos et al. propose a profit maximization approach based on microeconomics to direct the allotment of cloud resources for consumption in highly scalable master-worker virtual infrastructure [2]. Their proposed approach can maximize per-user financial profit and achieves resource sharing proportional to each user's budget. Zhang and Ardagna

propose a resource allocation controller for autonomic computing data center environments which maximizes the provider profits associated with multiclass Service Level Agreements [7]. Their main focus is on controlling the request routing and the processing sharing scheduling policies under predefined classes of jobs and utility-wise step functions. These approaches do not capture the differentiation of willingness-to-pay between clients and service demands are considered all equally, which is not realistic for IAAS environments described in this paper.

VM placement and migration approaches focusing on power optimization and profit maximization have received significant attention too and a number of related approaches have been proposed to address this issue [8–11]. Reference [12] proposes and evaluates energy-aware allocation policies that aim to maximize the average revenue received by the provider per unit time. A bin-packing based approach to maximize the resource satisfaction, minimize the number of application starts and stops, and balance the load across machines on a given datacenter is presented in [13]. The main focus of this work is on prompt response to changes in application demands but the problem of revenue issue is not investigated.

7. Conclusion and Future Work

In this paper, we argue that flexible provisioning of batch type job execution services in IAAS environments raises challenges not addressed by prior work on provisioning raw machine resources. We formulate provider's revenue maximization as an optimization problem, show its NP hardness, and develop a metaheuristic solution based on genetic algorithm. We illustrate the significance of proposed approach on a simulated environment. The experimental simulations and numerical results demonstrate that our proposed approach finds out near-optimal solutions to the revenue maximization problem under different problem sizes. The experimental results also show that the proposed GA optimization approach is sound on both efficiency and effectiveness for solving ICPRM problem.

For the future work, there are more related research issues worth investigating. First, we would like to extend our results to study solicitation of honest utility functions reporting by mechanisms design. Second, we are also interested in exploring pricing issues with multiple cloud providers, such as competition and peering.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

The authors would like to thank the anonymous referees for their insightful comments that have improved the quality of the paper. This work is supported in part by the National Natural Science Foundation of China (61402263), the Natural Science Foundation of Shandong Province (ZR2014FQ031), and the Science & Technology Development Projects of Shandong Province (2014GGH201007), China.

References

[1] L. M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner, "A break in the clouds: towards a cloud definition," *ACM SIGCOMM Computer Communication Review*, vol. 39, no. 1, pp. 50–55, 2008.

[2] K. Tsakalozos, H. Kllapi, E. Sitaridi, M. Roussopoulos, D. Paparas, and A. Delis, "Flexible use of cloud resources through profit maximization and price discrimination," in *Proceedings of the IEEE 27th International Conference on Data Engineering (ICDE '11)*, pp. 75–86, IEEE, April 2011.

[3] S. Martello and P. Toth, "Algorithms for knapsack problems," *Surveys in Combinatorial Optimization*, vol. 31, pp. 213–258, 1987.

[4] B. Urgaonkar and P. Shenoy, "Sharc: managing CPU and network bandwidth in shared clusters," *IEEE Transactions on Parallel and Distributed Systems*, vol. 15, no. 1, pp. 2–17, 2004.

[5] X. Jin, Y.-K. Kwok, and Y. Yan, "Competitive cloud resource pricing under a smart grid environment," in *Proceedings of the 5th IEEE International Conference on Cloud Computing Technology and Science (CloudCom '13)*, pp. 655–662, IEEE, December 2013.

[6] S. Chaisiri, B. Lee, and D. Niyato, "Competitive cloud resource pricing under a smart grid environment," in *Proceedings of the 9th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology*, pp. 1–4, 2012.

[7] L. Zhang and D. Ardagna, "SLA based profit optimization in autonomic computing systems," in *Proceedings of the 2nd International Conference on Service Oriented Computing (ICSOC '04)*, pp. 173–182, ACM, New York, NY, USA, November 2004.

[8] B. Li, J. Li, J. Huai, T. Wo, Q. Li, and L. Zhong, "EnaCloud: an energy-saving application live placement approach for cloud computing environments," in *Proceedings of the IEEE International Conference on Cloud Computing*, pp. 17–24, IEEE, September 2009.

[9] A. Beloglazov and R. Buyya, "Energy efficient resource management in virtualized cloud data centers," in *Proceedings of the 10th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing (CCGrid '10)*, pp. 826–831, May 2010.

[10] A. Beloglazov, J. Abawajy, and R. Buyya, "Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing," *Future Generation Computer Systems*, vol. 28, no. 5, pp. 755–768, 2012.

[11] S. Chaisiri, B.-S. Lee, and D. Niyato, "Optimal virtual machine placement across multiple cloud providers," in *Proceedings of the IEEE Asia-Pacific Services Computing Conference (APSCC '09)*, pp. 103–110, IEEE, Singapore, December 2009.

[12] M. Mazzucco, D. Dyachuk, and R. Deters, "Maximizing cloud providers' revenues via energy aware allocation policies," in *Proceedings of the 3rd IEEE International Conference on Cloud Computing*, pp. 131–138, IEEE, July 2010.

[13] C. Tang, M. Steinder, M. Spreitzer, and G. Pacifici, "A scalable application placement controller for enterprise data centers," in *Proceedings of the 16th International Conference on World Wide Web*, pp. 331–340, ACM, May 2007.