J. Soc. Korea Ind. Syst. Eng Vol. 38, No. 2 : 91-100, June 2015
http://dx.doi.org/10.11627/jkise.2015.38.2.91
ISSN : 2005-0461(print)
ISSN : 2287-7975(online)

An Efficient Two-Phase Heuristic Policy for Acceptance Control in IaaS Cloud Service

Moon Kyung Kim* · Jin Young Choi**†

*Hyundai AutoEver · Department of Industrial Engineering
**Ajou University

IaaS 클라우드 서비스 수락제어를 위한 효율적인 2단계 휴리스틱 정책

김문경* · 최진영**†

*현대 오토에버
**아주대학교 산업공학과

In this study, we propose an efficient two-phase heuristic policy, called an acceptance tolerance control policy, for Infrastructure as a Service (IaaS) cloud services that considers both the service provider and customer in terms of profit and satisfaction, respectively. Each time an IaaS cloud service is requested, this policy determines whether the service is accepted or rejected by calculating the potential for realizing the two performance objectives. Moreover, it uses acceptance tolerance to identify the possibility for error with the chosen decision while compensating for both future fluctuations in customer demand and error possibilities based on past decisions. We conducted a numerical experiment to verify the performance of the proposed policy using several actual IaaS cloud service specifications and comparing it with other heuristics.

Keywords : Acceptance Tolerance, Service Provider Profit, Customer Satisfaction, Inter-arrival time, Cloud Computing, IaaS

1. Introduction

Cloud computing, which expands the concept of utility computing, has recently attracted considerable attention in the IT industry. In the Cloud, users are not required to own IT resources such as a CPU, memory, storage, and bandwidth. Instead, the Cloud is an on-demand online computing system that is available at all times [1], [2]. Because large investments or upgrades on IT equipment and facilities are unnecessary, Cloud can reduce costs and increase business productivity. Examples of effective cloud services are the Amazon Web Service [3], IBM SmartCloud [4], and Microsoft Azure [5].

Cloud services can be categorized into three types [6] : Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). Among these, IaaS offers virtualized computing resources such as a CPU, memory, storage, and servers. Amazon Web Services [3] and the online backup solution Mozy [7] are examples of IaaS. Resource management procedures for an IaaS cloud service are as follows. When a cloud service is requested from a customer, a service level agreement (SLA) resource allocator determines whether to accept it, and then a service manager

Received 14 April 2015; Finally Revised 8 May 2015;
Accepted 8 May 2015
† Corresponding Author : choijy@ajou.ac.kr

controls the selection, allocation, and schedule of resources necessary to provide the service accepted. The price of the requested service is managed by the pricing and accounting divisions of the cloud service, and the usability of the resource is verified by the virtual machine monitor. The cloud service is then offered to the user, and the virtual machine allocated to the accepted service begins to operate [8]. However, because the cloud service is based on an SLA between the service provider and customer, the service provider is responsible to provide a service that is efficient and satisfies the SLA. This is accomplished through the use of available dynamic virtual resources to construct a virtual content delivery network [9, 10]. An extensive literature review on the construction and components/dimensions of cloud computing service can be found in [11].

Most studies on the admission control in a cloud service have focused on means of maximizing profit by the service provider's efforts and operations in managing revenue. Anandasivam [12] calculated the bid price through a simulation in which a genetic algorithm is employed before a service request is accepted. The bid price represents the potential value of a service and depends on the amount of remaining resources. By comparing the bid price with the service price, the system determines whether the service requested is acceptable. Puschel et al. [13] evaluated an acceptance control algorithm that manages revenue by examining a real-world gaming system. Moaker et al. [14] considered an auction-based mechanism that determines service acceptance in continuous queries, and introduced a decision policy and pricing mechanism to maximize profits. Toosi et al. [15] used cloud federation to propose a service acceptance control algorithm that produces IaaS-level cloud services. However, most of these studies designed policies that only consider the profit of the service provider, and not customer satisfaction. This could lead to admission control systems accepting only expensive service requests. Therefore, a customer's satisfaction would diminish as the number of rejected service requests increase, thus necessitating an arbitration method to balance them efficiently.

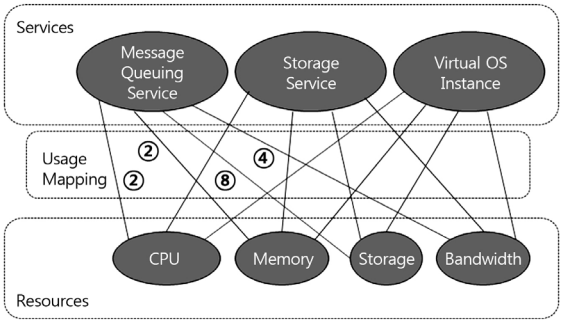
Based on these deficiencies in research and potential pitfalls, we propose an efficient two-phase acceptance tolerance control policy (ATCP) for IaaS cloud services that considers both service provider profit, and customer satisfaction. Each time the cloud service is requested, the service provider determines whether that service is to be accepted or declined by calculating the potential of the aforementioned performance objectives. Furthermore, compensating for future fluctuations in customer demand and error possibility based on past decisions is possible. This is accomplished by using allowable tolerance error to determine the possibility for error in choosing whether to accept or reject the cloud service. To assess the performance of ACTP, we conduct a numerical experiment by using several real IaaS cloud service specifications. Our results reveal its outstanding performance compared to other heuristics.

The remainder of this paper is organized as follows. Section 2 defines the research problem and assumptions regarding the decision-making processes of a cloud service. Section 3 develops the criteria for the acceptance of the requested service and proposes the two-phase ATCP. Section 4 describes a numerical experiment conducted to verify the performance of the proposed policy. Finally, Section 5 discusses the significance of this study and possible directions for future research.

2. Problem Definition and Assumptions

2.1 Problem Definition

This study examines IaaS-level basic cloud services provided in service environments, as shown in <Figure 1> [1]. IaaS can be formally described as follows. A service provider offers a cloud service $i \in \{1, \cdots, n\}$ by using certain resources such as CPU, memory, storage, and bandwidth, as well as specified quantities of each. Each resource $h \in \{1, \cdots, m\}$ is a quantifiable and separate unit. For example, one unit of CPU refers to a 1.25 GHz virtual CPU and one unit of memory or storage means 1 GB. Service i uses an a_{hi} amount of resource h and consists of $(a_{1i}, a_{2i}, \cdots, a_{mi})$ amounts of each resource. For instance, the message queuing service shown in <Figure 1> uses two units of CPU and includes



<Figure 1> Example of IaaS

two units of memory, eight units of storage, and three units of bandwidth. In addition, service i has a fixed price r_i, representing the amount of profit for a single unit of service i being accepted and sold. Each resource h initially has a capacity of C_h and is consumed by a_{hi} whenever the requested service i is accepted. Then, whenever the request for service i occurs at time t, the service provider must determine whether the service requested is accepted by considering the remaining quantity of each resource h. This is represented as $C_h - \overline{C}_{ht}$, where $\overline{C}_{ht}$ is the amount of resource h consumed from the beginning of the service to the time t.

<Table 1> Example Specification of IaaS

Resource	Services					Capacity (Units)
	1	2	3	4	5	
CPU(units)	2	4	3	8	10	1,500
Memory(units)	2	8	3	4	10	1,500
Storage(units)	8	2	4	4	8	1,500
Bandwidth(units)	4	2	8	4	8	1,500
Price($)	18	19.5	22.5	27	46	

<Table 1> shows example specifications for IaaS provided in a cloud service environment [17]. In the second row of the table, Services 1 to 5 $(n=5)$ are applicable to IaaS services shown in <Figure 1> such as the message queuing service, storage service, and virtual instance. In the first column, there are four resources $(m=4)$ that can be used to construct these five kinds of cloud services. As shown in the last column, each resource has the same limited capacity of $C_h = 1{,}500$, $\forall h$. The last row shows the price that the service provider can charge for each accepted service. Finally, each middle cell represents the amount of each resource consumed by a specific service. For example, Service 1 uses two units of CPU $(a_{11}=2)$, two units of memory $(a_{21}=2)$, eight units of storage $(a_{31}=8)$, and four units of bandwidth $(a_{41}=4)$. If Service 1 is accepted at a time t, all the relevant resources are reduced by those amounts. Thus, during the service sales period in which services are sold, the service provider faces the problem of whether or not to accept a requested service in order to maximize total profit. In addition, the service provider must consider the customer's satisfaction.

Based on the operational behavior of IaaS previously described, in this study, we aim to develop an efficient service policy for decision-making regarding the acceptance of IaaS-level services requested in a cloud service environment. We emphasize two performance objectives : service provider profit, and customer satisfaction.

2.2 Assumptions

In providing an IaaS cloud service, we assume the following.

- The service provider provides an IaaS by using the limited capacity of cloud resources.
- At a time t, the maximum number of service requests is one.
- Cancelation of accepted service orders is not allowed.
- If even one type of resource is completely consumed, the IaaS service is no longer provided.
- All accepted services are provided for a given period of time such that the price for service i is constant, which reflects its usage time.
- The service sales period T is sufficiently long for selling all types of services.
- The inter-arrival time of service requests is constant or exponentially distributed.

3. Two-phase Acceptance Tolerance Control Policy

3.1 Rules for Accepting a Service Request

We develop an acceptance criterion (i.e., one that supports the service provider's decision) for a service request in a cloud computing environment. The proposed criterion is based on an estimation of the potential value of resources required to provide service i that has been requested at time t. The potential value of resources for service i is evaluated by considering both the potential profit of the service provider, and the potential satisfaction of the customer by using relevant resources. This is further described as follows.

First, for service i requested at time t, we define the potential profit of the service provider using relevant resources as the potential maximum profit (PMP), which can be formulated as a linear programming model in LP1.

$$(LP1)\ PMP_t = Max \sum_{j \neq i} r_j x_j \quad (1)$$

$$s.t. \quad \sum_{j \neq i} a_{hj} x_j \leq a_{hi},\ h = 1, \cdots, m \quad (2)$$

$$x_j \geq 0,\ j = 1, \cdots, n, \quad (3)$$

where $x_j(j=1, 2, \cdots, n)$ is the number of acceptable customers for service j, that will be used in the following linear programming models. This formulation computes the maximum profit obtainable by using $(a_{1i}, a_{2i}, \cdots, a_{mi})$ when providing services other than service i.

It is clear that if the optimal objective value of LP1, or PMP_t, is greater than the price of service i, then declining the corresponding request for service i is recommended. However, because of the dynamic behavior of service requests received, no guarantee exists that we will obtain the profit of PMP_t after rejecting service i. Therefore, accepting service i may be reasonable if the difference of $PMP_t - r_i$ is not greater than a threshold value. We set the threshold to $\alpha \times 100\%$ of r_i so that α can then represent the error tolerance of the service provider's decision regarding its potential profit. This implies that the service provider does not reject service i if PMP_t is not greater than $(1+\alpha) \times r_i$. From the viewpoint of the service provider, this criterion can produce a possible error of $\alpha \times 100\%$ with respect to r_i by accepting service i in the event that $PMP_t - r_i \leq \alpha \cdot r_i$. The value of α can be any non-negative value depending on the tendency of the service provider to taking a risk. For example, if the service provider has a conservative tendency, then the value of α can be set as $0 \leq \alpha < 1$. Otherwise, $\alpha \geq 1$.

Based on this analysis, the first criterion for considering the potential profit of the service provider when accepting the requested service i at time t is defined as follows.

Decision rule 1 : Service i requested at time t is accepted only if it satisfies $PMP_t \leq r_i(1+\alpha)$, where $\alpha(\geq 0)$ represents the error tolerance of the service provider's decision regarding its potential profit.

Second, regarding customer satisfaction, we can consider the maximum number of accepted customer requests by using available resources. When an IaaS service is initially launched at initial time $t=0$, the maximum number of acceptable customers can be calculated by solving the following linear programming problem.

$$(LP2)\ K = Max \sum_{j=1}^{n} x_j \tag{4}$$

$$s.t. \quad \sum_{j=i}^{n} a_{hj} x_j \leq C_h,\ h=1, \cdots, m \tag{5}$$

$$x_j \geq 0,\ j=1, \cdots, n, \tag{6}$$

where Equation 4 is the number of all acceptable customer requests, and Equation 5 describes the limited conditions of each resource. The optimal solution $X^*_{Lp_2} = (x^*_1, x^*_2, \cdots, x^*_n)$ of LP2 indicates that a maximum of K requests can be accepted by using all available resources, with maximum x^*_j customer requests for each service $j \in \{1, 2, \cdots, n\}$.

However, in order to achieve $X^*_{Lp_2} = (x^*_1, x^*_2, \cdots, x^*_n)$ in a real situation, the service provider may purposely have to accept or decline many customer service requests, because of the dynamic behavior of service requests received. In terms of customer satisfaction, this may not be acceptable because many customer requests may then be rejected, resulting in many customer complaints. Moreover, although it may be favorable to the service provider if its decisions result in a value equal or nearly equal to $X^*_{LP_2}$, regarding the accepted number of customers, no guarantee exists that $X^*_{Lp_2} = (x^*_1, x^*_2, \cdots, x^*_n)$ can maximize the service provider's profit.

Based on these observations, for service i requested at time t, we evaluate the potential customer satisfaction using relevant resources and by determining whether rejecting service i can help to provide an even greater number of customer services while also reducing the possibility of customer requests being denied. We incorporate these considerations into our decision criterion to pursue the following : (i) maximizing the number of customer requests that can be accepted, and (ii) minimizing the number of customer requests that can be rejected, both using relevant resources. By considering these two objectives, we expect to eliminate scenarios in which a large number of customer requests are purposely declined to achieve $X^*_{LP_2}$. We can also encourage the acceptance of as high a number of customer requests as possible.

In order to pursue the first objective, we define the potential maximum number of customers (PMNC) using $(a_{1i}, a_{2i}, \cdots, a_{mi})$ for service i requested at time t as the maximum number of acceptable customer requests using relevant resources rather than accepting service i, which can be formulated by the following linear programming.

$$(LP3)\ PMNC_t = Max \sum_{j \neq 1} x_j \tag{7}$$

$$s.t. \quad \sum_{j \neq i}^{n} a_{hj} x_j \leq a_{hi},\ h=1, \cdots, m \tag{8}$$

$$x_j \geq 0,\ j=1, \cdots, n, \tag{9}$$

As in the case of PMP_t, it is clear that we can reject service i if $PMNC_t > 1$. However, that we will have $PMNC_t$ customer requests in the future is not guaranteed. This means that a decision must be made to accept service i if the difference $PMNC_t - 1$ is not greater than a threshold value β, where β represents the error tolerance of the service provider decision regarding customer satisfaction. This criterion can be interpreted as allowing the service provider to produce a possible error of $\beta \times 100(\%)$ with respect to one unit of service i by accepting service i for a case in which $PMNC_t - 1 \leq \beta$. The value of β can be any non-negative value depending on the inherent tendencies of the service provider. For example, if the service provider has a conservative tendency, then $0 \leq \beta < 1$. Otherwise, $\beta \geq 1$.

Based on this result, we establish the second decision-making criterion for potential customer satisfaction regarding the number of accepted customers as follows.

Decision rule 2 : Service i requested at time t is accepted only if it satisfies $PMNC_t \leq 1 + \beta$, where $\beta(\geq 0)$ is the error tolerance of the service provider's decision regarding the customer's satisfaction.

However, for the second objective we must also consider that too many customer requests may be rejected based on Decision Rules 1 and 2. Specifically, if the requested service i consists of many resources, then we can honor more than one customer request, and Decision Rule 2 $PMNC_t \leq 1 + \beta$ will be easily violated when using reasonable values of β, thus resulting in the rejection of service i. This result might lead us to a local feasible solution whereby we accept only those services that use few resources. Therefore, we need a method to prevent the second rule from using this localization as a solution, which can be performed by allowing the decision rules to diversify in the early stage of the sales period.

We design this method by having the service provider accept the rejected service i requested at time t with probability $1 - e^{-t}$, even if it has been rejected by either Decision Rule 1 or 2. By then generating a random number γ between 0 and 1, the method accepts the service only if $1 - e^{-t} < \gamma$. This acceptance probability is a decreasing function of t, meaning that the service provider expresses more confidence in the results of decision rules as time passes. Based on this method, we establish the last rule for minimizing the number of customer requests that have been rejected based on customer satisfaction.

Decision rule 3 : Although service i requested at time t will be rejected by either Decision Rule 1 or 2, it is finally accepted if it satisfies $P(acceptance) = 1 - e^{-t} < \gamma$, where γ is a randomly generated number between 0 and 1.

Consequently, when the service provider decides to accept service i requested at time t, it can determine both the service provider profit and customer satisfaction by applying these three decision rules sequentially.

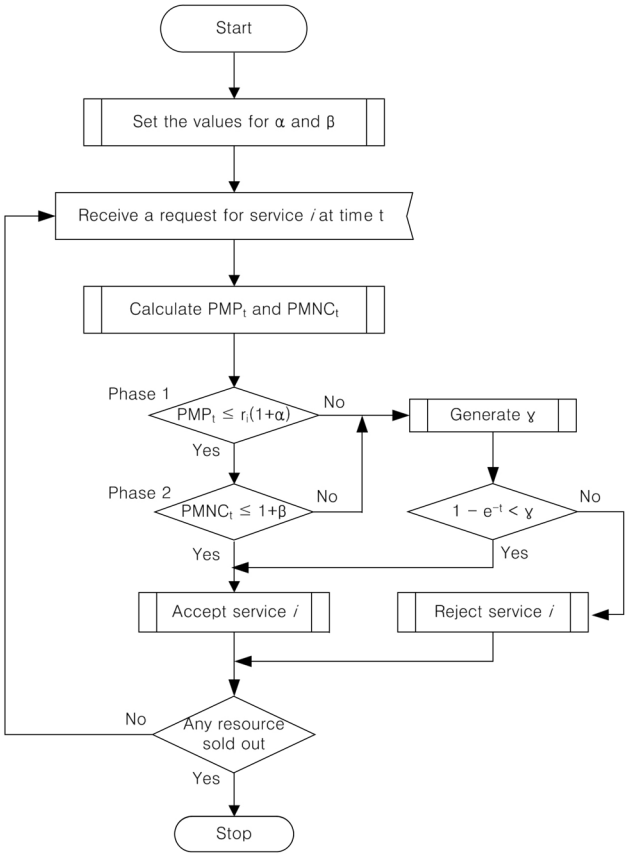
3.2 Design of Two-Phase ATCP

Based on the decision rules previously described, we design an efficient two-phase ATCP. <Figure 2> shows a flow chart of this policy, and the algorithm operates as follows.

Whenever an IaaS cloud service receives a new request for service i, the ATCP computes PMP_t and $PMNC_t$, then compares them with the relevant values during the following two steps. In Phase 1, the ATCP tests Decision Rule 1. If the rule is violated, then the ATCP generates a random number γ and applies Decision Rule 3 for the purpose of exploration. Otherwise, the ATCP progresses to Phase 2, in which it tests Decision Rule 2 and follows the same procedure as in Phase 1. If Decision Rule 3 is not satisfied during Phase 2, then the requested service i is rejected.

Using a given set of parameter values α and β, this procedure can be repeatedly applied to a series of service requests. We continue to compute PMP_t and $PMNC_t$, and determine whether each service request is to be accepted or rejected. The procedure ends when at least one resource is consumed, and it is not possible to provide any additional service. This is the overall procedure of the two-phase ATCP for the two-objective resource provisions that we suggest.

The performance of the policy depends on the values of α and β, which can be chosen by the service provider to control the precision of the policy. This can be done by performing numerical experiments with different values. Specifically, for different pairs of α and β, we can generate a series of service requests and then apply the two-phase ATCP. When the algorithm stops, we can compute two performance metrics (the service provider profit and the number of rejected customer requests) and identify appropriate values of α and β. However, the values to be set depend on the problem setting, such as the number of service types for IaaS, the capacities of resources, and the resource usage for each service.



<Figure 2> Flow Chart of the Proposed Two-Phase ATCP

4. Numerical Experiment

4.1 Design of the Experiment

In order to evaluate the proposed two-phase ATCP, we conducted a numerical experiment. First, we considered three virtual machine services currently provided in an actual cloud environment as follows.

- IBM SmartCloud Enterprise service [4]
- Amazon EC2 instance service [3]
- Gogrid Cloud Server service [16]

<Tables 2>~<Table 4> list the specifications of these three cloud IaaS services. The categories in the tables are the same as those in <Table 1>. However, each of these cloud services provides services with different names. For example, IBM's SmartCloud Enterprise provides levels of service that it groups into copper, bronze, silver, gold, and platinum, which are based on the quantity of consumed resources. These resources include the CPU, memory, and storage. Without loss of generality, each resource is assumed to have a limited capacity.

<Table 2> Specification of IBM SmartCloud Service

Service / Resource	Copper	Bronze	Silver	Gold	Platinum	Capacity (Units)
CPU	2	2	4	8	16	20,000
Memory	4	4	8	16	16	40,000
Storage	60	850	1,024	1,024	2,048	3,000,000
Price($)	0.115	0.160	0.200	0.320	0.630	

<Table 3> Specification of Amazon EC2 Service

Service / Resource	Small	Medium	Large	Extra Large	Capacity (units)
CPU	1	2	4	8	30,000
Memory	1.7	3.75	7.5	15	60,000
Storage	160	410	850	1,690	6,000,000
Price($)	0.115	0.230	0.460	0.920	

<Table 4> Specification of Gogrid Cloud Service

Service / Resource	S1	S2	S3	S4	Capacity (units)
CPU	1	2	4	8	30,000
Memory	1	2	4	8	30,000
Storage	50	100	200	400	1,500,000
Price($)	0.12	0.24	0.36	0.48	

We assumed that these services are offered in the same environment as given in <Figure 1>, and that service providers want to maximize profit and ensure customer satisfaction. Each time a customer then requests a service, the service provider may offer various services that share common resources by using the two-phase ATCP. To represent different behavior of service requests, we considered two types of models for the inter-arrival time of service requests as follows.

(i) *Constant inter-arrival time model* : Customer service requests occurr in a constant time interval $(t=1)$, with the occurrence probability of each service distributed either uniformly or inversely proportional to the service price.

(ii) *Exponential inter-arrival time model* : Customer requests for each service occurr according to an exponential distribution, with the rate distributed either uniformly or inversely proportional to the service price.

<Table 5>~<Table 7> show the occurrence probability (or rate) of service requests for the three IaaS services considered in our experiment. Regarding the constant inter-arrival time model, each value listed represents the probability that a service request at each time belongs to a certain type of service. However, in the case of the exponential inter-arrival time model, the numbers indicate the arrival rates of different services, meaning the number of requests for the corresponding service per unit time. Moreover, the values in the second row in each table correspond to a uniform case meaning equal probabilities or equal arrival rates. The third row shows the non-uniform values of probabilities, or arrival rates that are inversely proportional to the service price. This can be computed as follows. First, we calculate the inversed values of price for each service. We then normalize them by making their sum equal to 1, representing probabilities or arrival rates.

<Table 5> Occurrence Probability(rate) for IBM SmartCloud Service

Type \ Service	Copper	Bronze	Silver	Gold	Platinum
Uniform	0.2	0.2	0.2	0.2	0.2
Non-uniform	0.35	0.25	0.2	0.13	0.07

<Table 6> Occurrence Probability(rate) for Amazon EC2 Service

Type \ Service	Small	Medium	Large	Extra Large
Uniform	0.25	0.25	0.25	0.25
Non- uniform	0.53	0.27	0.13	0.07

<Table 7> Occurrence Probability(rate) for Gogrid Cloud Service

Type \ Service	S1	S2	S3	S4
Uniform	0.25	0.25	0.25	0.25
Non-constant	0.48	0.24	0.16	0.12

<Table 8> Acceptance Tolerance of Services Considered

	α	β
IBM'sSmartCloud Enterprise service	0.7	0.2
Amazon's E2C instance service	0.2	0.2
Gogrid's Cloud Servers service	0.4	0.2

We implemented the proposed algorithm by using Visual Studio 2008 C++. Through experiments, the total cloud service sales period T was determined to be $T = 40{,}000$ for IBM's service and $T = 130{,}000$ for Amazon's and Gogrid's services. The proper values of α and β for the two-phase ATCP algorithm were determined through repeated experiments, as presented in <Table 8>.

The performance of the proposed algorithm was evaluated based on the service provider profit and the customer's satisfaction. We evaluated customer satisfaction by computing the customer rejection ratio. Thus, the two performance measures (PMs) were defined as follows :

- *PM1-Service provider profit* : the total profits of the services accepted during the service sales period T.
- *PM2-Customer rejection ratio* : the ratio of the number of service requests rejected during the service sales period T(i.e., the number of service requests rejected/The total number of service requests).

Moreover, we compared the performance of the proposed algorithm with existing two heuristics : (i) the first-come-first-serve (FCFS), which is the most representative dispatching rule currently known, and (ii) the customized bid price policy (CBPP), which was proposed to maximize service provider profit [12]. To the best of our knowledge, CBPP is the most recently developed heuristic revealing the best performance at maximizing service provider profit. When a new service request is received, CBPP uses data acquired from the history of service sales to calculate the bid price. The bid price represents the potential value of the service to the service provider. This number changes depending on the amount of remaining resources. Thus, it accepts the service request only when the service price is less than or equal to the bid price.

4.2 Analysis of Experimental Results

We performed a numerical experiment by running the two-phase ATCP 30 times with each of the three types of IaaS services listed in <Table 2>~<Table 4>. <Table 9>~<Table 11> show the results of the experiment for PM1 and PM2. Cases 1 and 2 in the tables indicate the probability or arrival rate that is distributed uniformly and that is inversely proportional to the service price, respectively.

Regarding PM2, FCFS accepts service requests according to the order of their receipt. Therefore, no services are re-

jected and the value of PM2 becomes 0 in all cases. The values for PM1 and PM2 in the tables are based on the average yielded from 30 runtimes. Detailed experimental results for each model of inter-arrival times can be summarized as follows.

A : Case of the constant inter-arrival time model

- The values of the PM1 were higher when using the two-phase ATCP than when using either FCFS or CBPP for all three services. In the case of IBM and Gogrid services, the profits generated by the two-phase ATCP were higher than those generated by CBPP by approximately 40% for all cases of probability distributions. However, no considerable difference between FCFS and CBPP was apparent, with FCFS having a slightly higher PM1 value than did CBPP.
- Concerning the type of probability distribution (uniform or not), all heuristics had similar PM1 values for the uniform and non-uniform cases, thus indicating that the PM1 value does not depend on the type of probability distribution. Based on this observation, we can assume that the PM1 value converges to a specific value, particularly when the two-phase ATCP is used with constant inter-arrival time service requests.
- For PM2, FCFS yielded no rejection. However, the two-phase ATCP had approximately 7~45% lower PM2 values than did CBPP for all three services. This means that CBPP rejects more service requests than does the two-phase ATCP. Specifically, in Case 2 of Amazon's service, CBPP rejected more than 92% of the requested services. This produced a maximum difference of approximately 45% compared to results of the two-phase ATCP.
- CBPP showed no considerable differences in the two cases of probability distribution (uniform or not). However, when the two-phase ATCP is used, Case 1 produced a higher PM2 value by approximately 22~27% than did Case 2 for all three services. This result shows that the two-phase ATCP is effective at reducing PM2 in non-uniform probability distribution for service requests.

B : Case of the exponential inter-arrival time model

- The results were very similar to those generated by the constant inter-arrival time model. Specifically, the two-phase ATCP produced a higher PM1 than did the two heuristics (FCFS, CBPP) for all three services. In the case of PM2, the two-phase ATCP produced lower values by 10~46% than did CBPP, showing that CBPP rejected more service requests.
- It seems that the exponential inter-arrival time model

<Table 9> Experimental Results of IBM SmartCloud Service

	Constant inter-arrival time model						Exponential inter-arrival time model					
	PM1($)			PM2(%)			PM1($)			PM2(%)		
	ATCP	FCFS	CBPP	ATCP	FCFS	CBPP	ATCP	FCFS	CBPP	ATCP	FCFS	CBPP
Case 1	3449.41	3328.63	3266.37	74.96	0	90.99	3449.36	3229.82	3266.35	74.54	0	90.21
Case 2	3449.82	3440	3258.33	47.01	0	92.22	3449.82	3440	3257.94	47.48	0	93.90

<Table 10> Experimental Results of Amazon EC2 Service

	Constant inter-arrival time model						Exponential inter-arrival time model					
	PM1($)			PM2(%)			PM1($)			PM2(%)		
	ATCP	FCFS	CBPP	ATCP	FCFS	CBPP	ATCP	FCFS	CBPP	ATCP	FCFS	CBPP
Case 1	1286.67	850.71	787.93	71.26	0	87.73	1286.65	854.3	787.91	70.59	0	88.65
Case 2	1286.67	864.23	817.01	49.61	0	76.60	1286.67	862.81	819.98	48.48	0	74.52

<Table 11> Experimental Results of Gogrid Cloud Service

	Constant inter-arrival time model						Exponential inter-arrival time model					
	PM1($)			PM2(%)			PM1($)			PM2(%)		
	ATCP	FCFS	CBPP	ATCP	FCFS	CBPP	ATCP	FCFS	CBPP	ATCP	FCFS	CBPP
Case1	3599.37	2401.48	1832.54	74.99	0	82.83	3599.37	2408.68	1798.33	74.47	0	84.41
Case2	3599.76	2702.89	2167.39	47.13	0	83.71	3599.63	2688.29	2167.85	47.46	0	84.42

changes probabilistically the length of the inter-arrival times for consecutive service requests and does not affect the performance of the service policies considered in this study.

- Overall, the type of occurrence probability (or rate) of the requested service has a greater effect on the performance level of the resource provisioning policy for IaaS in cloud computing than does the length of the intervals between service requests.
- Moreover, for each applied heuristic, PM1 seems to gravitate to a specific value, and this value may be useful in identifying a boundary value for achieving an optimal PM1 for each service structure.

5. Conclusion

In this study, we proposed an efficient two-phase ATCP for IaaS cloud service with limited cloud resources. The policy determines the acceptance of service requests by considering both (i) the service provider profit and (ii) customer satisfaction. Specifically, we proposed two decision rules that evaluate the value of resources required for a requested service by computing the potential profit of the service provider and the potential satisfaction of the customer. Furthermore, we improved the performance of the suggested policy by removing from the effect of localization on the search procedure. By using the acceptance tolerance α and β, the algorithm reflects the error possibility of the acceptance decision, while compensating for both future fluctuations in customer demand and the error possibilities generated from past decisions.

The performance of the two-phase ATCP was tested by a numerical experiment using actual IaaS cloud service specifications. We considered two types of service requests with constant and exponential distribution. By defining two PMs (PM1 and PM2), the results showed that the two-phase ATCP performs extremely well compared to similar and representative heuristics. Furthermore, we observed that the type of occurrence probability of the requested service, rather than the length of the intervals between service requests, has a greater effect on the performance of the resource provisioning policy for an IaaS cloud service. We expect that these results may be used as the baseline for developing an efficient service operation or resource utilization policy, which can be extended to other cloud services such as PaaS and SaaS. This is one direction for future studies.

Acknowledgement

This work was supported by Defense Acquisition Program Administration and Agency for Defense Development under the contract UD140022PD, Korea.

References

[1] Buyya, R., Yeo, C.S., Venugopal, S., Broberg, J., and Brandic, I., Cloud computing and emerging IT platforms : Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation Computer Systems*, 2009, Vol. 25, No. 6, pp. 599-616.

[2] Rodero-Merino, L., Vaquero, L.M., Galan, V.G.F., Fontan, J., Montero, R.S., and Llorente, I.M., From infrastructure delivery to service management in Clouds. *Future Generation Computer Systems*, 2010, Vol. 26, No. 8, pp. 1226-1240.

[3] Huckman, R.S., Pisano, G.P., and Kind, L., Amazon web service. Harvard Business School Case (609-048), 2008.

[4] IBMSmartCloud, Accessed March 2014 http://www-935.ibm.com/services/us/en/cloud-enterprise/index.html.

[5] WindowsAzure, Accessed March 2014, http://www.windowsazure.com/en-us.

[6] Voorsluys, W., Broberg, J., and Buyya, R. *Introduction to Cloud Computing*, John Wiley and Sons, Inc, 2011.

[7] mozy, Accessed March 2014, http://mozy.com/.

[8] Yoon, C.H., Technical trends in managing cloud computing resources, Grid Middleware Research Center KAIST, 2011, pp. 9-11.

[9] Wu, L., Gang, S.K., and Buyya, R., SLA-based admission control for a Software-as-a-Service provider in Cloud computing environments. *Journal of Computer and System Sciences*, 2012, Vol. 78, No. 5, pp. 1280-1299.

[10] Um, T.W., Lee, H., Woo, R., and Choi, J.K., Dynamic resource allocation and scheduling for cloud-based virtual content delivery networks. *ETRI Journal*, April 2014, Vol. 36, No. 2, pp. 197-205.

[11] Jula, A., Sundararajan, E., and Othman, Z., Cloud computing service composition : A systematic literature review. *Expert Systems with Applications*, 2014, Vol. 41, pp. 3809-3824.

[12] Anandasivam, A., Consumer Preferences and Bid-Price

Control for Cloud Services, doctoral dissertation, Institute for Information Systems and Management, University of Karlsruhe, 2010, pp. 96-100.

[13] Puschel, T., Lang, F., Bodenstein, C., and Neumann, D., A Service Request Acceptance Model for Revenue Optimization-Evaluating Policies Using a Web Based Resource Management Game. 43rd Hawaii International Conference on System Science, 2010.

[14] Moakar, L.A., Chrysanthis, P.K., Chung, C., Guirguis, S., Labrinidis, A., Neophyton, P., and Pruhs, K., Admission Control Mechanisms for Continuous Queries in the Cloud. 26th IEEE International Conference on Data Engineering, 2010.

[15] Toosi, A.N., Calheiros, R.N., Thulasiram, R.K., and Buyya, R., Resource Provisioining Policies to Increase IaaS Provider's Profit in a Federated Cloud Environment. IEEE International Conference on High Performance Computing and Communications, 2011.

[16] Gogrid, Accessed March 2014, http://www.gogrid.com/.

[17] Anandasivam, A., Buschek, S., and Buyya, R., A Heuristic Approach for Capacity Control in Clouds. IEEE Conference on Commerce and Enterprise Computing, 2009.

ORCID

Moon Kyung Kim | http://orcid.org/0000-0002-3834-6280

Jin Young Choi | http://orcid.org/0000-0001-6397-3107