Cluster Comput (2017) 20:3537–3549
DOI 10.1007/s10586-017-1148-1

Adaptive resource provisioning method using application-aware machine learning based on job history in heterogeneous infrastructures

Jieun Choi[1] · **Yoonhee Kim**[2]

Received: 27 February 2017 / Revised: 23 June 2017 / Accepted: 23 August 2017 / Published online: 4 September 2017

Abstract With the remarkable growth in cloud computing, computing resources can be manipulated on-demand in most scientific fields. This enables scientists to strategically select their experimental environment. Since it is hard to offer cloud resources in accordance with application characteristics, efficient resource provisioning methods are needed. This paper proposes an adaptive resource provisioning method using an application-aware machine learning technique that is based on the job history in heterogeneous infrastructures. The proposed resource provisioning method is built on two main concepts. First, it provides application-aware resource provisioning through the profiling of scientific application in a heterogeneous computing infrastructure. A resource provisioning model uses the resource usage profiles of scientific applications and job history data in heterogeneous computing infrastructures. In addition to the multilayer perceptron machine learning method, an error back-propagation approach is applied to analyze job history to re-learn the error of the output value. Second, it offers an adaptive resource scaling that is invoked by the availability of resource changes. An adaptive resource management method results in an enhancement of the overall balance between the performance and utilization of a system. For the experiments with the two CPU-intensive applications according to the method, a heterogeneous infrastructure comprising clusters and cloud environments is used. Experimental results indicate that the use of the proposed method can gratify user requests (cost and execution time) regarding its application and enhance resource usage effectiveness.

Keywords Adaptive provisioning · Multilayer perceptron · Application-aware · Auto-scaling · Heterogeneous infrastructure

✉ Yoonhee Kim
yulan@sookmyung.ac.kr

Jieun Choi
jieun1205@kisti.re.kr

[1] National Institute of Supercomputing and Networking, KISTI, Daejeon 34141, Korea

[2] Department of Computer Science, Sookmyung Women's University, Seoul 140-742, Korea

1 Introduction

Cloud computing appears as a new computing paradigm that uses virtualization technology and allows pay-per-use computing services. It provides scalable and virtualized large-scale IT resources (server, storage, software, and network) as services [1,2]. With the development of cloud computing, it has become possible to utilize computing resources as needed in various scientific fields. Increasingly, heterogeneous computing infrastructures are being deployed that integrate cloud environments into existing computing infrastructures such as clusters and grids in multiple data centers [3–6]. Alternatively, it is difficult for cloud providers to meet their needs and provide the resources they need, so they entrust users with the task of making of wholly-needed resource choices. Limits exist, however, regarding the support of diverse scientific applications that require high performance computing (HPC) and high throughput computing (HTC) in which only the user-selected resources are used; therefore, there is a need to develop a resource provisioning technique that pre-selects resources that are suitable for the nature of scientific applications and computing environments and makes these available to users. At present, studies on resource provisioning techniques in which various statistical analysis and inference methods are used have been carried out; however, existing

methods did not consider the characteristics of scientific applications and did not target heterogeneous computing infrastructures in data centers that use heterogeneous computing resources. Therefore, in the heterogeneous computing infrastructure environment, including cloud, resource provisioning and resource management techniques that can maximize the benefit between the user and the provider are needed.

This paper proposes an adaptive resource provisioning method in which an application-aware machine learning technique that is based on the job history in heterogeneous infrastructures is employed. The proposed resource provisioning method is built on two main concepts. First, it provides application-aware resource provisioning through profiling of the scientific application in the heterogeneous computing infrastructure. A resource provisioning model uses resource usage profiles of the scientific application and job history analysis in heterogeneous computing infrastructures comprising cluster and cloud environments. In addition to multilayer perceptron (MLP) machine learning methods, an error back-propagation technique is used to analyze the job history to re-learn the error of the output value. Secondly, it offers an adaptive resource scaling that is invoked by the availability change of the resources. An adaptive resource management method results in the enhancement of the overall balance between the performances and the utilization of the system. The proposed method maximizes the benefit between the user and the provider by considering the characteristics of the scientific application, the requirements of the user, and the state of the system for the resource provisioning technique. In addition, adaptive resource management techniques that dynamically adjust the number of resources according to systemic changes enable an effective computing infrastructure management.

In this study, two CPU-intensive applications were investigated according to the proposed model and algorithms in a heterogeneous infrastructure comprising cluster and cloud environments. Experimental results show that the use of the proposed method could satisfy user requests (cost and execution time) regarding their application and improve resource usage efficiency. Based on the proposed job history learning model, the superiority of the proposed method was demonstrated using inferred resources and a performance comparison was performed in which the inferred resources used other policies. Through the proposed algorithm, it was possible to efficiently provide resources in terms of cost and performance by inferring suitable resources for the performance of applications and in consideration of user requirements and usable systemic resources; furthermore, it was possible to effectively manage resources in response to changes in the available resources of the system.

The major contents and methods of this study are as follows:

- Presents a job history learning model using an MLP and error back-propagation technique for learning the working history of computational science applications in a heterogeneous computing infrastructure.
- Creates input data and output data of the suggested learning model using profiling data of computational science application and the result of the work performed in the heterogeneous computing infrastructure comprising a cluster (SGE: Sun Grid Engine [7]) and a cloud (OpenStack [8]), and learning to verify the job history learning model is developed.
- Suggests an adaptive cloud provisioning service system and algorithm that can dynamically scale resources according to systemic changes.
- Verifies the performance of the proposed algorithm through comparative real experiments and simulation.

The rest of this paper is structured as follows. Section 2 discusses related studies, while Sect. 3 introduces the adaptive resource provisioning model based on the job history learning technique. In Sect. 4, adaptive resource provisioning algorithms are discussed in detail, while the experiments and results are presented in Sect. 5. Section 6 concludes the paper and discusses proposed future research.

2 Related studies

With the development of cloud computing, it has become possible to utilize computing resources as needed in various scientific fields. Increasingly, heterogeneous computing infrastructures that integrate cloud environments into existing computing infrastructures such as clusters and grids are being deployed in multiple data centers [3–6]. For the HPC communities (DIRAC [9], Condor-G [10], gUSE/WS-PGRADE [11], and HTCaaS [12,13]), the cloud was recently integrated with conventional distributed computing environments such as the grid, cluster, and desktop grid. A few studies addressed the need for the simultaneous control of heterogeneous computing infrastructures.

Mateescu et al. [3] presented the concept of the ElastiCluster, in which conventional HPC, grid, and cloud computing are combined to achieve effective and predictable executions of HPC workloads. In [3], the authors noted that no single infrastructure is the best solution from all points of view. Moca et al. [4] described a fault-tolerant and trust-aware scheduler, which allows for the execution of Bag-of-Tasks (BoTs) applications on elastic and hybrid distributed computing infrastructures. Delamare et al. [5] completed the best effort as part of the EUs European Desktop Grid (EDGI) FP7 project by creating a fast and reliable technique to meet the expected completion time of BoTs applications in

distributed computing infrastructures. Delamare's team proposed the SpeQuloS service that provides cloud resource dynamically. The distributed infrastructure with remote agent control (DIRAC) [9] is a software framework for distributed computing that provides access to heterogeneous computing infrastructure required by users. It integrates various grid, cloud, and cluster resources, provides concurrent interactions, and enables stable use of resources. Condor-G [10] combines technologies from the Condor project and the Globus project to provide a robust job-broker function and is widely used in grid environments; it matches the job requirements according to the characteristics of the resource, and supports grid, cloud, and cluster resources. gUSE/WS-PGRADE [11] provides a workflow-driven scientific application execution interface, as well as an integrated grid, a cloud, a cluster, and a desktop grid. In a previous study [6], the authors investigated the job distribution ratio in a heterogeneous infrastructure using high-throughput computing as a service (HTCaaS) service type. With accelerated increase in the use of cloud infrastructure, a corresponding increase in dynamic resource provisioning techniques provides the required resources. In this paper, a resource provisioning model is proposed based on application characteristic profiles and job history analysis in a heterogeneous computing infrastructure comprising of cluster and cloud environments.

In this section, different related studies and systems are described with a concentrate on resource provisioning. In addition, related research on adaptive resource provisioning and management techniques in which various machine learning and reasoning methods were used are compared with our study.

2.1 Application-aware resource provisioning method based reasoning technique

Cloud providers entrust users with wholly-needed resource choices, since it is difficult for cloud providers to meet their needs and provide the resources they need. However, there are limits to supporting diverse scientific applications that require high-performance and high-throughput computing with only user-selected resources; therefore, a resource provisioning technique that pre-selects resources that are suitable for the nature of scientific applications and computing environments, and provides them to users is needed. There are a few studies on resource provisioning techniques in which various statistical analysis and inference methods were used.

Kim et al. [14] proposed a fuzzy logic based resource evaluation method for provisioning scheduling in a cloud environment. In [14], performance states that are hard to define, such as CPU utilization, RAM utilization, and Net I/O of a virtual machine (VM), were analyzed, and were used in resource availability evaluation. The algorithm deduces the state of the VM using static attributes, such as CPU utilization, main storage utilization, and network latency, and extrapolates the availability using the current VM state, the work processing performance, and the number of jobs in the queue; when a job is submitted, it selects a VM with the highest availability. This, however, does not indicate that the appropriate VM for the application was used in such a way that only the availability of the last used VM is inferred. Rao et al. [15] proposed a distributed self-learning scheme for resilient provisioning of cloud resources that can satisfy the SLA while increasing the resource utilization rate; however, this is not suitable for high-performance computing environments for the implementation of high-performance and high-processing science applications in which the system uses the users feedback rather than self-learning, depending on the characteristics of the application. Tesauro et al. [16] proposed a hybrid reinforcement learning scheme in which offline reinforcement learning and queuing models were used for automatic resource allocation. Their scheme represents a way to solve the problem of performance decline of online reinforcement learning of their previous research, and regarding the offline reinforcement learning method, the optimum policy in the dynamic environment is found through the system state and various actions. Meanwhile, HTTP requests are distributed according to a queuing theory, and the performance is maintained through the expected response time. Sukhija et al. [17] proposed a portfolio-based dynamic loop scheduling algorithm in which supervised learning was used in heterogeneous computing environments. This is a method for finding an optimal scheduling technique for scientific applications in distributed and parallel environments according to the MLP learning method in which the characteristics of the application, system state, and performance were considered. However, this method is only used for grid computing resource and a cloud environment in which on-demand allocation of resources is not considered.

The above studies consist of resource provisioning techniques that did not consider the characteristics of scientific applications and did not target heterogeneous infrastructure of a data center that uses various heterogeneous computing resources. Therefore, it is necessary to select a resource that is suitable for the required application performance while maximizing the benefit between the user and the provider, and in which the characteristics of the scientific application are considered through the analysis of the job history. In this study, a neural network learning method based on a multilayer perceptron is used for the analysis of job histories, while application characteristics are considered, and it is possible to provide the resources that are suitable for the system while satisfying user requirements and application performance.

2.2 Adaptive resource management

The cloud environment has advantages of scalability and accessibility. Therefore, a resource management that adapts to user requirements and resource changes in the cloud environment has been actively studied [17–21]. Typical adaptive methods include the auto-scaling method comprising horizontal and vertical scaling. Horizontal scaling is a scaling in/out method that reduces or increases the number of virtual machines allocated based on the amount of work or used resources. Vertical scaling is a method of scaling up/down the amount of resources (CPU, Mem, and Disk) allocated to one virtual machine [22]. Amazon provides CloudWatch [18] services that monitor cloud resources and running applications, collects resource usage indicators, and provides horizontal scaling based on configured thresholds. In the openstack, groups and policies can be created for virtual machine scaling using Heat components, and an auto-scaling service for horizontal scaling according to attribute settings such as the maximum and minimum sizes of the scaling group is provided.

Studies [18–21] on the auto-scaling techniques have been conducted through a prediction of future resource utilization of the application. Samuel et al. [19] introduced a pre-prediction model for future resource requirements of Web applications to satisfy the SLA in a cloud environment and proposed a resource provisioning method based on this. The response time and throughput of the Web service were used as the SLA, and three types of machine learning techniques, namely, linear regression, neural network, and support vector machine (SVM), were used as prediction models; here, resource usage and scales of the virtual machine were predicted accordingly. Similarly, Bashar et al. [20] used a Bayesian network based prediction model, and Nikravesh et al. [21] compared auto-scaling with CloudWatch [18] based on a predictive model in which the hidden Markov model (HMM) was used. In a previous study [23], the optimal resource allocation ratio according to the characteristics of the application in the cluster and the cloud environments was found, and the result of the application of the auto-scaling technique according to delay occurrence was analyzed; here, the technique prevents the SLA from violating horizontal scaling when a deadline violation of the application due to the delay is detected. In [23], two types of scientific applications (BoTs, workflows) were simulated in a heterogeneous environment. The auto-scaling technique was applied to control the number of virtual machines such that they do not violate the SLA deadline and systemic performance can be guaranteed; however, this did not consider system conditions for changes in the available resources. The above adaptive resource provisioning techniques determine the scaling of resources based on a predictive model or policy. In this paper, an adaptive resource provisioning scheme is proposed based on the application characteristics and analysis of the job history, and in which both horizontal and vertical scaling are used.

3 Resource provisioning model using application-aware machine learning based on job history

Machine learning is a process in which the system learns to improve its performance from empirical data based on interactions with the external environment, even if it is not explicitly programmed [24,25]. Machine learning is divided into *supervised learning*, *unsupervised learning*, and *reinforcement learning*, depending on the type of data required for learning. Among these learning types, map learning presents learning data comprising an input/output pair, and a function map where learning data is learned. Such guidance learning is suitable for pattern recognition, classification, prediction, and regression analysis of data [23]. Alternatively, various learning model structures, such as the neural network, SVM, decision tree, and Bayesian network exist depending on the function expression method for the performance of machine learning [25]. Among these, the neural network is processed by the activation function in which input and connection weights are used, and the output can be changed depending on the selected activation function. Perceptron learning is a learning algorithm of the representative neural network structure, which is divided into single-layer perceptron or MLP learning, depending on the layer. The MLP maps a set of input data onto a set of appropriate outputs, and it consists of multiple layers of nodes (neuron) in a directed graph with an activation function. MLP is a model that can solve a non-linear separation problem (XOR), which is impossible to solve using single-layer perceptron.

In this section, the application-aware resource provisioning model is introduced. To design this model based on job history data, items of the job history data must be defined. The job history data is composed of four items as presented in Table 1.

First, profiling information of the application represents the characteristics of the scientific application that the user wants to submit to the system to perform the application. The average CPU usage and the average memory usage require pre-profiling data for the application. Second, the system state should use information about the system, and it reflects the systemic situation through the current usage according to each computing resource. In this paper, the focus on is two clustering and cloud heterogeneous computing infrastructures. Third, VM information is information about the type of VM used to perform the operation. Lastly, the execution history information includes the execution time of the task and the total cost of cloud resource usage. In the case

Table 1 Job history items

Categories	Items
Application profiles	Name of application
	Type of application
	Average CPU utilization
	Average memory utilization
	Number of tasks
	Size of input files
System status	System specification
	Current usage of resource
VM information	vCPU
	Memory
	Disk
	Flavor name
	Flavor cost
	Number of used VM
History data	Total job execution time
	Total cloud resource cost
	Job distribution ratio

of the total job share in the execution history, the ratio of the tasks assigned to the cluster is expressed in Eq. 1 as follows:

$$0 \leq w_t \leq 1, \sum_0^i w_i = 1 \tag{1}$$

For example, in an environment that targets clusters and cloud computing infrastructures, w_0 is a cluster and w_1 is a cloud. When 10,000 tasks are being performed, if w_0 is 0.6, and 6000 tasks are performed on the cluster resource, then 4000 tasks are performed on the cloud resource. The sum of w_0 and w_1 is always 1.

The job history data that is composed of various items shows a linear non-separable data distribution. (The data distribution figure according to job history items is given in [26]). Therefore, in this study, a learning model in which an MLP model that can be applied to linear inseparability problems is used to investigate history learning of the scientific application.

The MLP is composed of an input layer X_i, a hidden layer Z_i, and an output layer Y_i, and it is composed of a weight and a threshold vector (θ). The input values in the learning model are set as the application profiling of the job history item, the system state, and the execution history, and the output value is VM information. To apply the error back-propagation technique, it is necessary to subject the activation function of the MLP to the following differential sigmoid function. The input value (Input (Z_i)) of the i-th neuron of the hidden layer is calculated using the presented input vector, and the output value (Output (Z_i)) of the hidden layer is calculated using the sigmoid function. The output of the hidden layer is used to calculate the input value (Input (Y_i)) of neuron i of the output layer. Similarly, the output value (Output (Y_i)) is calculated using the sigmoid function. The error back-propagation technique updates the weight by re-learning the error of the expected output value. The weight of the hidden layer takes the error calculated in the next layer (output layer) and multiplies it by the weight to calculate the error, since the target value is not known by the hidden layer.

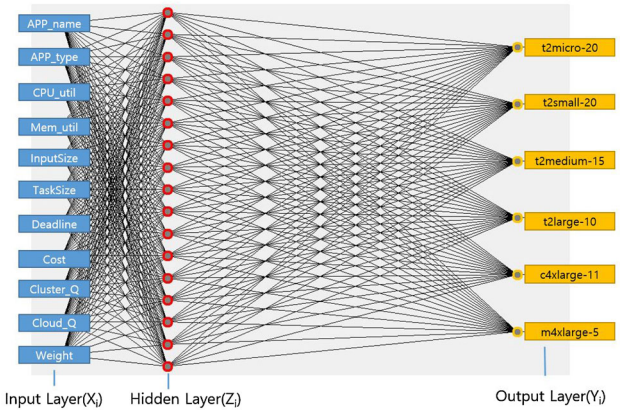

Fig. 1 Resource provisioning model based on job history using MLP

Table 2 Validation results for the proposed resource provisioning model

Correctly classified instances	7 (77.7778%)
Incorrectly classified instances	2 (22.2222%)
Kappa statistic	0.7188
Mean absolute error (MAE)	0.1073
Root mean squared error (RMSE)	0.2398

Figure 1, in which the MLP learning technique was used, describes the proposed resource provisioning model based on the job history in terms of Weka [27]. Input data for the model involves historical data, system status, and application profiles, while the output is VM information of the job history items. Input data for the learning stage of the learning model used 156 data, and input data for the verification stage of the learning model used 9 data. The learning factor was worked out several times and was adopted as a parameter in the case showing the highest exactness as follows: learning rate = 0.2, momentum = 0.8, number of learning = 500, number of allowed consecutive errors = 20, number of hidden layers = 17, number of input layers = 11, and number of output layers = 6. In the prediction attempt of the verification stage, the exactness of the learning model was calculated. Table 2 presents validation results for the proposed resource provisioning model. The proposed adaptive resource provisioning model showed an exactness of 77.7778%. The kappa statistic indicates a substantial value of 0.7188 [28]. The value of the mean absolute error was approximately 10%. Assuming that

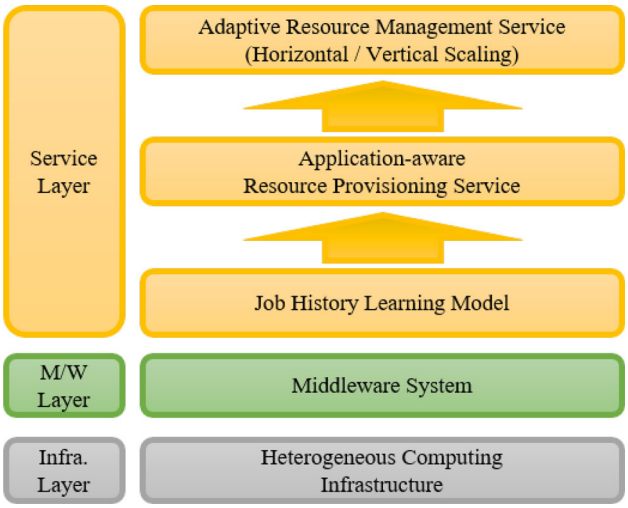

Fig. 2 Service architecture model

Table 3 Notations

Notation	Description
$\mathbb{R}$	A user request
P	A user policy for SLA
SP	Scaling policy of the system
PD	Profiling data for $\mathbb{R}$
R_i	A resource
	$R_i \in \{Cluster, Cloud\}$
SR_i	Current status of R_i
TR_i	Type of resource R_i
UR	Resource types are selected from user
CSS	Current system status
$\|cR\|$	Number of changed resources
$\|rR\|$	Number of reduced / released resources
MID	Input data of job history learning model

the errors that were generated follows a normal distribution, more than 68% did not have a larger error rate than the root mean squared error (RMSE) and less than 95% had an error rate less than 2*RMSE.

Figure 2 shows a sketch of a service architecture model for an adaptive resource provisioning method in which the job history learning technique is used in a heterogeneous infrastructure. Essentially, it consists of the following three layers: Infrastructure layer, Middleware layer, and Service layer. In this study, the heterogeneous computing infrastructure was targeted; therefore, clusters and cloud environments were deployed on the HTCaaS middleware. The Service layer mostly consists of the following two services: the Application-aware Resource Provisioning Service (ARPS) based on the job history learning model and the Adaptive Resource Management Service (ARMS). In the ARPS, when a user presents an SLA (cost and deadline) and scientific applications to the system through the middleware, virtual machines are assigned according to suitable SLA and system status based on deductions from the job history learning model. While executing scientific applications using deduced resources, the ARMS monitors usable resources to supply services and adjust the number of virtual machines according to system status.

4 Adaptive resource provisioning method

In this section, four algorithms that can provision resources in heterogeneous infrastructure, monitor the systemic availability ratio, and adjust the VM according to the system state are introduced. In this study, the existence of job profiling information to be performed by a user is assumed. Furthermore, an installed in advance image can be used for resources, such as a library and a compiler, which are necessary for the performance of scientific application in terms of task performance that can occur in a VM. The presumptions and notations of the proposed algorithms are presented in Table 3.

Algorithm 1 Adaptive Resource Provisioning Algorithm

Input: a Request $\mathbb{R}$, a user policy P, user resources UR
1: **Submit** $\mathbb{R} = \{ App_{name}, InputFile_{size}, |InputFile|, SLA_d, SLA_c \}$
2: Default SF, HJS, DV , SD = false ;
3: **while** $\mathbb{R}$ **do**
4: **Set** $PD = \{ \mathbb{R}, App_{type}, CPU_{util}, Mem_{util} \}$
5: **Set** $CSS = \{ SR_i \mid i = 0,1,\cdots, N-1, R_i \in UR \}$
6: **ResourceProvisioning**(PD, CSS) ;
7: **AvailabilityMonitoring**(***interval***) ;
8: **end while**

Algorithm 1 describes an adaptive resource provisioning algorithm in which a user presents a job request, $\mathbb{R}$, a user policy, and preferred resource types. The default values of the system failure (SF), higher priority job submission (HJS), user-defined deadline violation (DV), and scaling decision (SD) are false (line 2). A system failure means that the state of the resource changes from available to unavailable. At every monitoring interval, algorithm 1 calls the Availability-Monitoring function which monitors the state of the resource (line 7). In lines 3 and 4, the profiling data (PD) and the current system status (CSS) are set. Algorithm 2 is called by using the set PD and CSS(line 6).

Algorithm 2 shows a resource provisioning that is based on application characteristic profiles and job history analysis in the heterogeneous computing infrastructure, including cloud environments. In algorithm 2, a value for the learning model input data is set by using the PD and the CSS (line 1); here, it detects a appropriate VM type and number in accor-

dance with the MID by using the MLP model (line 2). If the resource is a cloud, the learning model based VM provisioning result is selected (lines 4–6). Meanwhile, other resources are provisioned by the number of available resources (lines 7–9).

Algorithm 2 Resource Provisioning Algorithm

Input: Profiling data PD, Current system status CSS
1: **Set** $MID = \{ PD, CSS \}$
2: VM_{type} and $|VM| \leftarrow MLP(MID)$;
3: **if** $R_i ==$ Cloud **then**
4: $TR_i \leftarrow VM_{type}$;
5: $|R_i| \leftarrow |VM|$;
6: **else**
7: $TR_i \leftarrow R_i$;
8: $|R_i| \leftarrow AvailableR_i$;
9: **end if**
Output: Resource Provisioning RP
$= \{(TR_i , |R_i|) \mid i = 0,1,\cdots, \mathrm{N}-1, R_i \in UR\}$

Algorithm 3 Monitoring Algorithm

Input: a result of resource provisioning RP , a user policy P, monitoring interval *interval*
1: **while** true **do**
2: $SF, R_i \leftarrow DetectSF()$;
3: $HJS \leftarrow DetectHJS()$;
4: **if** $SF == true$ **then**
5: $|cR_i| \leftarrow (|R_i| - |rR_i|)$;
6: RESCHEDULE($waitingJob, cR_i$);
7: $DV \leftarrow CompareEFT(EFT, SLA_d)$;
8: **end if**
9: **if** $HJS == true$ **then**
10: **Increase** $JobPriority$;
11: **Release** $|rR|$;
12: $|cR_i| \leftarrow (|R_i| - |rR_i|)$;
13: RESCHEDULE($waitingJob, cR_i$),;
14: $DV \leftarrow CompareEFT(EFT, SLA_d)$;
15: **end if**
16: **if** $DV == true$ **then**
17: **if** $P == PERFORMANCE$ **then**
18: $SD \leftarrow true$;
19: **else**
20: $Update(SLA_d)$;
21: **end if**
22: **end if**
23: **Return** Scaling Decision SD, $|rR_i|$
24: **Sleep** *interval*
25: **end while**

Algorithm 3 presents available resources monitoring algorithm. Algorithm 3 monitors the systemic failure, higher priority job submissions, and user-defined deadline violations at every monitoring interval (lines 2 and 3). If a systemic failure occurs, the amount of available resources is reduced depending on the number of failed resources and the waiting jobs of the malfunction resources are rescheduled to the changed resources cR_i (lines 4–8). If the new job priorities are higher than that of the current job, the priority of the latter is increased and the size of resources to release is calculated (lines 9–11). The amount of available resources is reduced in accordance with rR and the holding jobs are rescheduled to cR_i (lines 12 and 13). Despite the rescheduling, if a deadline contravention is discovered, then a scaling decision is settled in accordance with user policy. If the user policy is the performance, the scaling decision is true and algorithm 4 is executed. If the user policy is the cost, the user deadline is updated with the calculated application execution time (lines 17–21). Owing to the execution of algorithm 3 at every monitoring interval, algorithm 4 is executed only when the scaling decision (SD) is true.

Algorithm 4 Adaptive Resource Scaling Algorithm

Input: Scaling Decision SD, Scaling Policy SP, Amount of reduced resources $|rR_i|$,
1: **while** SD **do**
2: **Switch** (SP)
3: **case** $HORIZONTAL$**:**
4: $TR_{cloud} \leftarrow VM_{type}$;
5: $|R_i| \leftarrow |R_{cloud}| + |rR_i|$;
6: **case** $VERTICAL$**:**
7: $TR_{newcloud} \leftarrow NewVMtype(VM_{type})$;
8: $|R_j| \leftarrow |rR_i|$;
9: **end switch**
10: **end while**
Output: Adaptive Resource Provisioning ARP
$= \{(TR_i , |R_i|) \mid i = 0,1,\cdots, \mathrm{N}-1, R_i \in UR\}$

Algorithm 4 indicates the adaptive resource scaling method. It shows the flow of horizontal and vertical execution according to systemic scaling policy in a situation where scaling progress has already been determined by Algorithm 3. In horizontal scaling, the increasing amount of the current VM is computed (lines 3–5). Vertical scaling should calculate the modified VM size of the new VM (lines 6–9). The yield of algorithm 4 forms the results of the suggested adaptive resource provisioning (ARP) method.

5 Experiments

The experiments that were conducted to validate the proposed ARP method are presented in this section. First, the heterogeneous computing infrastructure is presented in Sect. 5.1, and the target application is described in Sect. 5.2. Subsequently, experimental results are presented in Sect. 5.3.

5.1 Heterogeneous computing infrastructures

HTCaaS [12] targets to accelerate the exploration of large scale HTC or MTC tasks for different computing environments such as grids, supercomputers, clusters, and clouds.

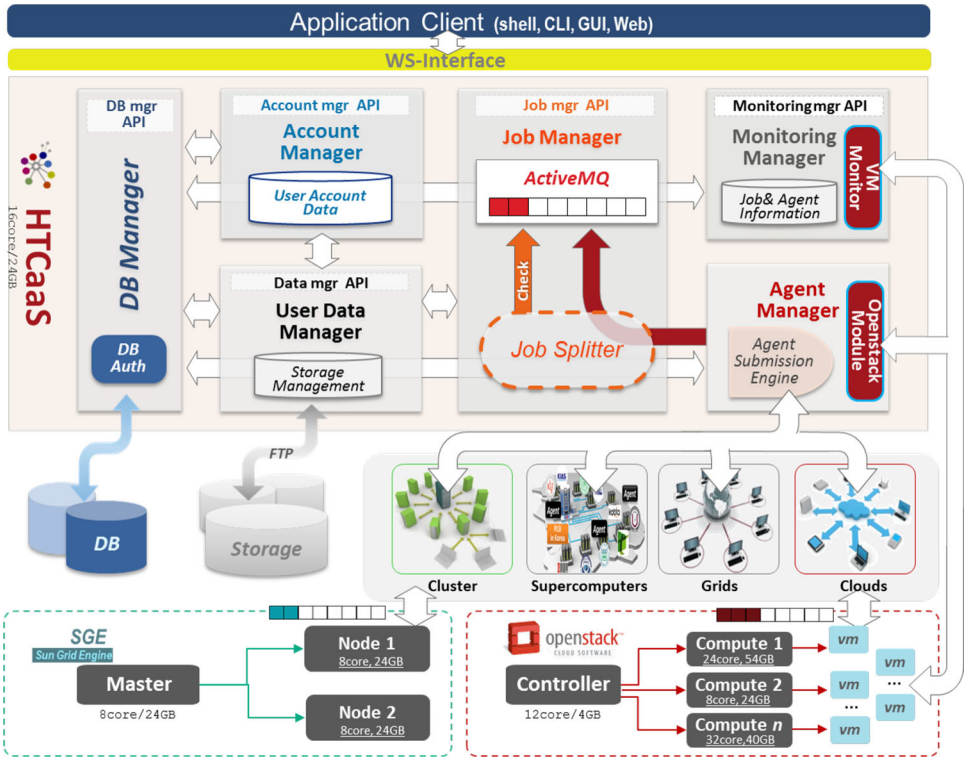

Fig. 3 Heterogeneous computing infrastructures for scientific applications using HTCaaS [6]

It can hide the diversity of the integration of heterogeneous resources from end users, and enables users to effectually present many tasks immediately. For HTCaaS, a pilot-based multi-level scheduling technique that assists the decoupling of resource assignment from resource binding is applied. The ordinary pilot (or agent) itself is a common batch job that is presented by the HTCaaS system and is allocated to resources by the local batch scheduler. Then, it executes the pulling and executing sub-jobs, as well as adjusting the launch and monitoring procedures (Fig. 3).

The computing resources consist of local cluster and private cloud resources in which HTCaaS is used. The local cluster uses a Sun Grid Engine (SGE) [7], which is a batch queuing system supported by Sun Microsystems. OpenStack [8] is an open source software that provides large pools of computing, storage, and networking resources that are used for cloud environments. The OpenStack cloud environment is composed of one Intel(R) Core(TM) i7-4930K CPU @ 3.40 GHz controller with 12 cores of CPU and 4 GB of RAM, one Intel(R) Core(TM) i7-4930K CPU @ 3.40 GHz compute node with 8 CPU cores and 24 GB RAM, and two Intel(R) Xeon(R) CPU E5-26500 @ 2.00 GHz compute nodes, with 24 CPU and 32 CPU cores and 40 and 54 GB RAM. In the experiments, six flavors, namely, t2.micro, t2.small, t2.medium, t2.large, c4.xlarge, and m4.xlarge, were used for the VM, and its cost was configured according to the Amazon EC2 [29] as presented in Table 4. Each VM was created identically using the Ubuntu 12.04 Server image.

Table 4 VM information

Name	vCPU	Mem (GB)	Disk (GB)	Cost ($) per hour
t2.micro	1	1	5	0.02
t2.small	1	2	5	0.04
t2.medium	2	4	5	0.08
t2.large	2	8	10	0.16
c4.xlarge	4	8	10	0.24
m4.xlarge	4	16	10	0.33

5.2 Target application

Computational Science (Scientific Computing) is a field wherein science and engineering problems are solved using numerical methods and computer calculations, and various phenomena are solved using computers [30]. In this study, a molecular docking application (Autodock3 [31]) from the field of drug development and physical particle generation application (Pythia [32]) from high-energy physics was used in the experiments.

In terms of a new drug development in computational biology, computer operated molecular docking simulations have been conducted to determine the as yet unknown protein lipid binding mode, such that research on the function of protein or the development of a new drug that inhibits the function of protein can be conducted. Molecular docking requires tremendous amount of computation because all possible con-

figurations must be investigated, and these molecular docking applications are CPU-intensive applications [33]. Molecular docking applications include DOCK and Autodock. In this study, Autodock3 was used from among various molecular docking algorithms. Autodock3 application generated profiling information while showing an average of 99.12% CPU usage and 0.7% memory usage, and each lipid molecular data became one input file. One input data size was approximately 0.2 MB, and the number of lipid molecules that are to be molecularly docked were 200 (404 KB), 400 (887 KB), 600 (1.33 MB), 800 (1.80 MB), and 1000.

High-energy physics field performs large volume Monte Carlo simulations for prediction and verification in which particle accelerator data are used [12]. The high-energy physics accelerator particle collision simulation analyzes the generated fragment by colliding high-speed particle with the target material, followed by its reconstruction, and this makes it possible to find the target material structure. In this study, resource usage profiling information was collected while the Pythia application was implemented with a high-energy particle collision simulation. The average Pythia CPU utilization in a pure CPU-intensive application was 94.59%. In the experiment, job histories of 1200, 1400, 1600, 1800, and 2000 tasks were used in the Pythia application.

5.3 Experimental results

5.3.1 Analyzing learning based resource provisioning results

In this section, the result from the use of the inferred resources through the proposed learning model is presented. Three scenarios were considered depending on the availability of the resources. In the first scenario, the availabilities of the cluster and cloud systems were 100%. In the second scenario, the availability of the cloud system was 50%. In the third scenario, the availability of the cluster system was 50%. The case where the inferred resource with the proposed learning based (LB) method was used was compared with the case where the virtual resource was selected in accordance with cost-minimum (CM) and performance-maximum (PM) policies.
CPU-intensive application: In the experiment environments, an Autodock application that docks 200 ligand files was run. The w_0 value was given as 0.3, the SLA_d was given as 3000 s, and the SLA_c was given as \$1.4. According to availability scenarios, the learning based resource provisioning method deduced 15 VMs of the t2.medium type in case 1, 10 VMs of the t2.large type in case 2, and 5 VMs of the m4.xlarge type in case 3 as appropriate resources. The CM policy used 20 VMs of the t2.micro type and the PM policy used 11 VMs of the c4.xlarge. Figure 4a, b show the overall execution time and throughput of the three resource availability systemic scenarios when Autodock application was run with the selected resources according to LB, CM, and PM policies. Results of a comparison of the LB method with the CM policy with respect to the three cases show that the former approach can increase the speed by approximately 35, 7.3, and 34%, respectively. Results of a comparison of the PM policy with the CM policy show that the PM policy resulted in enhancements of 42.19, 12.42, and 42.28%, respectively, in the three cases. The throughput values of the LB method were improved by 31.14, 7.03, and 30.16%, respectively, compared with those of the CM policy, but they were decreased by 11.03, 5.32, and 12.13%, respectively, compared with the throughput values of the PM policy.
Pure CPU-intensive application: Figure 4c, d show the overall execution time and throughput of the three resource availability systemic scenarios when the Pythia application was run with the selected resources in accordance with the LB, CM, and PM policies. When 1600 Pythia tasks were run in the given heterogeneous computing infrastructure, w_0 was given as 0.4, SLA_d was given as 650 s, and SLA_c was given as \$1.3. According to the availability scenarios, the learning based resource provisioning method deduced 10 VMs of the t2.large in cases 1 and 2, and 5 VMs of the m4.xlarge in case 3. The CM policy used 20 VMs of the t2.micro type and the PM policy used 11 VMs of the c4.xlarge as appropriate resources. Results of a comparison of the LB method with the CM policy with respect to the three cases show that the former approach can increase the speed by approximately 1.32, 0.6, and 0.1%. Results of a comparison of the PM policy with the CM policy show that the former policy enhanced the three cases by 3.61, 3.12, and 1.32%, respectively. The throughput values of the LB method were improved by 1.28, 1.52, and 0.16% compared with the CM policy, but they were decreased by 2.3, 2.48, and 1.16% compared with the PM policy.

The results indicate that in all the scenarios, the execution time was short and the throughput was high compared with the CM. However, a longer execution time and a lower throughput were shown compared with the PM policy. In Figure 4, all scenarios show that the proposed LB inferred resources were appropriate for each state when resource availability changes. It also inferred resources differently depending on the characteristics of Autodock and Pythia applications. This is because the items indicating the system status and the attributes of the application in the job history items were reflected in our machine learning model. When the learning model includes values of system status and application profiles for creating job history data, it can be observed that VM deduction was possible by recognizing these values. Therefore, the proposed LB was more cost effective than the PM policy and exhibited a more efficient performance than the CM policy because the inference was based on resource availability, and application resource consumption characteristics can be made possible.

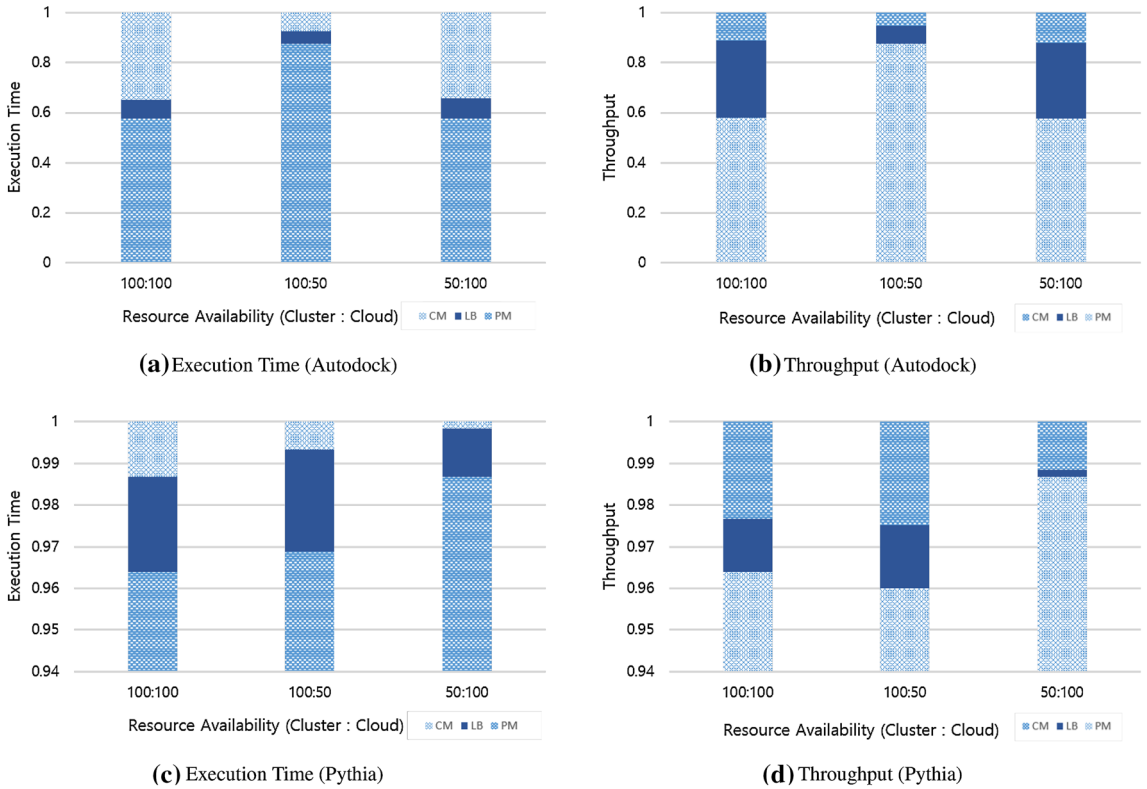

Fig. 4 Results of execution time and throughput in accordance with resource availability scenarios: **a**, **c** have been normalized with CM policy. **b**, **d** have been normalized with PM policy

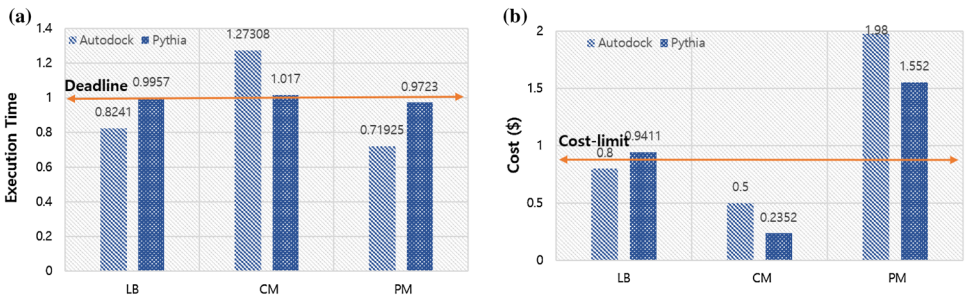

Fig. 5 Comparison of SLA satisfaction using LB, CM and PM policies

5.3.2 Performance comparison based on user requirements

Figure 5a, b show the SLA satisfaction performance during the operation of the Autodock and Pythia applications, respectively. The inferred resources according to the proposed learning based resource provisioning were compared with the CM (20 VMs of the t2.micro) and PM (11 VMs of the c4.xlarge) policies.

In the case of Autodock, the cost limit that based on the user SLA was $4.0, the user-defined deadline was 12,000 s, and the w_0 was 0.3. With respect to provisioned resources, 15 VMs of the t2.medium were deduced from the LB method. In the case of Pythia, the user-defined cost limit was $1.7, the user-defined deadline was 470 s, and the w_0 value was 0.4. In terms of selected resources, 10 VMs of the t2.large were inferenced from the learning based method.

These results show that the proposed method results in more money being spent compared with the CM policy. Alternatively, the execution time of the PM technique was significantly shorter than that of the proposed learning based resource provisioning, but the result of the PM was beyond the required cost. The CM technique showed results beyond the required deadline, and the PM technique exceeded the required cost. In Figure 5, all scenarios show that the proposed LB inferred resources appropriate for each state when the user-defined deadline and cost-limit were varied. It also inferred resources differently depending on the characteristics of Autodock and Pythia applications. This is because the items indicating history data in the job history items were

reflected in our machine learning model. When the learning model includes values of total job execution time and total cloud resource cost for creating job history data, it can be observed that VM deduction was possible by recognizing these values. The proposed LB can be more efficient than the CM and PM policies in terms of SLA satisfaction because it can implement reasoning that reflects user requirements for execution time and cost.

5.3.3 Adaptive resource provisioning performance analysis

To verify the performance of the adaptive resource provisioning scheme, simulation experiments were performed using CloudSim [34]. In the simulation, the Autodock application was considered. Based on real experimental data, 1000 tasks with average operation lengths of less than 1000 s were created. The monitoring interval was 10 s. The user-defined deadline was 200 min and the cost limit was $4.0. To demonstrate effective scaling, the user's policy was assumed as the performance. The performance of the adaptive resource provisioning scheme was analyzed with using 15 VMs of the t2.medium that were deduced by the job history learning model according to the proposed algorithm 2. The proposed ARP method in which both horizontal and vertical scaling were used was compared with the non-adaptive resource provisioning (NAP) method.

Change in resource availability: In the first scenario, some of the cluster resources changed to unavailable resulting in system failure. Figure 6 shows the number of total resources that were used and the total execution time. At 4000 s, the cluster system failed, and the number of resources was reduced from 24 to 15. Without ARP, the total number of resources was reduced to 30, the number of cloud resources was equal to 15, and the result was beyond the deadline. When ARP was used, the total number of resources was 39, and the number of cloud resources was increased to 24; but, the deadline was not overridden and the user requirements were satisfied.

Change in job priority: In the second scenario, another job with a higher priority was submitted during a job run and the job's priority was lowered. Figure 7 shows the number of total resources that was used and the total execution time in scenario 1. At 5000 s, a high-priority job was submitted and the system released 12 cluster resources. Without ARP, the total number of resources was reduced to 27 and the number of cloud resources was the same, but the result was beyond the deadline. When ARP was used, the number of resources was increased, but it did not exceed the deadline, indicating that user requirements have been met.

These results indicate that the suggested adaptive resource provisioning method is sensitive to changes in resource availability and job priorities and can satisfy user requirements. In addition, both horizontal and vertical scaling are sufficient to control resources of the cloud, but in the case of vertical scaling, it should be reflected to have some overhead including the VM reboot time. In the future, it is necessary to improve the algorithm such that the amount of resources to be controlled can be predicted and adjusted accurately.

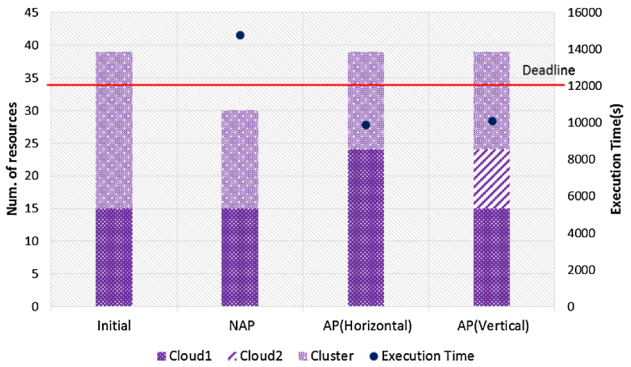

Fig. 6 Scenario 1: change in resource availability

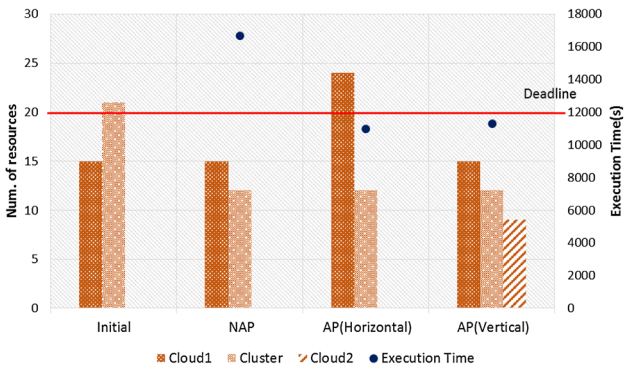

Fig. 7 Scenario 2: change in job priority

6 Conclusion

In this paper, an adaptive resource provisioning method that is suitable for application characteristics was proposed through the study of job histories in a heterogeneous computing environment in which clusters and cloud resources were used. For the job history learning model, the MLP and the error back-propagation approach were applied for the learning of job history of computational science applications in a heterogeneous computing infrastructure. In addition, an adaptive scheme in which a horizontal scaling that adjusts the number of virtual machines according to the system state was proposed. Two CPU-intensive applications were subjected to experiments according to the proposed model and algorithms in a heterogeneous infrastructure consisting of cluster and cloud environments. Through the proposed algorithm, it was possible to efficiently provision resources in terms of cost and performance by inferring appropriate resources in which user requirements and available systemic resources

were considered; furthermore, it was possible to effectively manage resources in response to changes in the available systemic resources. In this paper, the results that were mostly influenced by the number of VM vCPUs in the CPU-intensive applications were presented. In the future, we will carry out more experiments with diverse types of HPC and HTC applications. Furthermore, it will be necessary to conduct a performance comparison experiment with other inference techniques (related studies). Advanced adaptive algorithms are also needed to accurately predict the amount of resources. To do this, we will extend the job history data to conduct performance comparison experiments.

Acknowledgements This research was supported by the Next-Generation Information Computing Development Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Science, ICT & Future Planning (2015M 3C 4A7065646).

References

1. Lee, K., et al.: Standardization trends and strategies on cloud computing. Commun. Korean Inst. Inf. Sci. Eng. **28**(12), 27–33 (2010)
2. Lim, C.-S.: Cloud computing security technology. Rev. KIISC **19**(3), 14–17 (2009)
3. Mateescu, G., et al.: Hybrid computing where HPC meets grid and cloud computing. Future Gener. Comput. Syst. **27**(5), 440–453 (2011)
4. Moca, M., et al.: Multi-criteria and satisfaction oriented scheduling for hybrid distributed computing infrastructures. Future Gener. Comput. Syst. **55**, 428–443 (2016)
5. Delamare, S., et al.: SpeQuloS: a QoS service for BoT applications using best effort distributed computing infrastructures. In: Proceedings of the 21st International Symposium on HPDC, pp. 173–186 (2012)
6. Choi, J., et al.: A job dispatch optimization method on cluster and cloud for large-scale high-throughput computing service. In: IEEE International Conference on Cloud and Autonomic Computing, Cambridge, MA, USA, 21–24 September (2015)
7. SGE. http://www.oracle.com/technetwork/oem/grid-engine-166852.html
8. OpenStack. https://www.openstack.org/
9. DIRAC. http://diracgrid.org/files/docs/Overview/index.html?highlight=pilot
10. Condor, G.: A computation management agent for multi-institutional grids. Clust. Comput. **5**(3), 237–246 (2002)
11. gUSE/WS-PGRADE. http://guse.hu/about/home
12. Kim, S.K., et al.: HTCaaS (high throughput computing as a service) in supercomputing environment. J. Korean Contents Assoc. **14**(5), 817 (2014)
13. Hwang, S., et al.: HTCaaS: Efficient and simplified large-scale scientific computing over supercomputers, grids and cloud. The ACM Cloud and Autonomic Computing Conference Miami, Florida, USA, 59 August (2013)
14. Kim, J., et al.: Fuzzy logic-driven virtual machine resource evaluation method for cloud provisioning service. J. Korea Soc. Simul. **22**(1), 77–86 (2013)
15. Rao, J., et al.: A distributed self-learning approach for elastic provisioning of virtualized cloud resources. In: IEEE 19th Annual International Symposium on Modelling, Analysis, and Simulation of Computer and Telecommunication Systems, pp. 45–54 (2011)
16. Tesauro, G., et al.: A hybrid reinforcement learning approach to autonomic resource allocation: 2006 IEEE International Conference on Autonomic Computing, pp. 65–73 (2006)
17. Sukhija, N., et al.: A learning-based selection for portfolio scheduling of scientific applications on heterogeneous computing systems. J. Parallel Cloud Comput. **3**(4), 66–81 (2014)
18. Amazon CloudWatch. http://aws.amazon.com/cloudwatch/
19. Samuel, A.A., Akindele, B.A.: Proactive prediction models for web application resource provisioning in the cloud. In: International Conference on Transition from Observation to Knowledge to Intelligence, pp. 17–35 (2014)
20. Bashar, A.: Autonomic scaling of cloud computing resources using BN-based prediction models. In: 2013 IEEE 2nd International Conference on Cloud Networking, pp. 200–204 (2013)
21. Nikravesh, A.Y., et al.: Cloud resource auto-scaling system based on hidden markov model (HMM). In: 2014 IEEE International Conference on Semantic Computing, pp. 124–127, (2014)
22. Lorido-Botran, T., et al.: A review of auto-scaling techniques for elastic applications in cloud environments. J. Grid Comput. **12**(4), 559–592 (2014)
23. Choi, J., et al.: VVM auto-scaling methods for high throughput computing on hybrid infrastructure. Clust. Comput. **18**(3), 1063–1073 (2015)
24. Simon, P.: Too Big to Ignore: The Business Case for Big Data. Wiley, Hoboken (2013)
25. Zhang, B.T.: Next-generation machine learning technologies. Commun. Korean Inst. Inf. Sci. Eng. **25**(3), 96–107 (2007)
26. Choi, J.: An adaptive resource provisioning method based on application-aware job history learning technique, Masters Thesis, Sookmyung Womens University (2016)
27. Weka. http://www.cs.waikato.ac.nz/ml/weka/
28. Landis, J.R., et al.: The measurement of observer agreement for categorical data. Biometrics **33**(1), 159–174 (1977)
29. Amazon EC2. http://aws.amazon.com/
30. Scientific computing. https://en.wikipedia.org/wiki/Computational_science
31. Autodock. http://autodock.scripps.edu/
32. Sjstrand, T., Mrenna, S., Skands, P.Z.: PYTHIA 6.4 Physics and Ma. J. High Energy Phys. **2006**, 026 (2006)
33. Lee, K.W.: Practical introduction to computational biology. Korean. Biophys. Soc. Newsl. **10**(1), 10–22 (2004)
34. Calheiros, R.N., et al.: CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms, software: practice and experience, vol. 41, No. 1, pp. 23–50 (2011)

Jieun Choi is a researcher in National Institute of Supercomputing and Networking (NISN) at KISTI (Korea institute of Science and Technology Information). She received her B.S. and M.S. degree from Sookmyung Women's University in 2014 and 2016, respectively. Her research interests include cloud computing and management in distributed computing systems.

Yoonhee Kim is a professor in the Department of Computer Science, Sookmyung Women's University. She received her B.S. degree from Sookmyung Women's University in 1991, her M.S. degree and Ph.D. from Syracuse University in 1996 and 2001, respectively. She was a Research Staff Member at the Electronics and Telecommunication Research Institute during 1991 and 1994. Before joining the faculty of Sookmyung Women's University in 2001, she was on the faculty of the Computer Engineering Department at Rochester Institute of Technology in NY, USA. Her research interests span many aspects of runtime support and management in distributed computing systems.