An Allocation and Provisioning Model of Science Cloud for High Throughput Computing Applications

Seoyoung Kim
National Institute of Supercomputing and Networking, KISTI
Daejeon, Republic of Korea
sssyyy77@kisti.re.kr

Jik-Soo Kim
National Institute of Supercomputing and Networking, KISTI
Daejeon, Republic of Korea
jiksoo.kim@kisti.re.kr

Soonwook Hwang
National Institute of Supercomputing and Networking, KISTI
Daejeon, Republic of Korea
hwang@kisti.re.kr

Yoonhee Kim*
Dept. of Computer Science
Sookmyung Women's Univ.
Seoul, Republic of Korea
yulan@sookmyung.ac.kr

ABSTRACT

Recent cloud computing enables numerous scientists to earn advantages by serving on-demand and elastic resources whenever they desire computing resources. This science cloud paradigm has been actively developed and investigated to satisfy requirements of the scientists such as performance, feasibility and so on. However, effective allocation and provisioning virtual machines on clouds are still considered as a challenging issue in scientists using high throughput computing, since it determines whether they can earn benefits from economy of scale in clouds or not. Moreover, allocating the "right" provisioned cloud resources on an optimal data center is very important as performance can vary widely depending on where and under what circumstances it actually runs. In these reasons, it is required that an appropriate and suitable model for science cloud to support increasing scientists and computations.

In this paper, we present an allocation and provisioning model of science cloud, especially for high throughput computing applications. In this model, we utilize job traces where statistical method is applied to pick the most influential features for improving application performance. With the feature, the system determines where VM is deployed (allocation) and which instance type is proper (provisioning). An adaptive evaluation step which is subsequent to the job execution enables our model to adapt to dynamical computing environments. We show performance achievements as comparing the proposed model with other policies through experiments. Finally, we expect that improvement on performance as well as reduction of cost from resource consumption through our model.

*Corresponding Author

CAC '13, August 05 - 09 Miami, FL, USA

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems

General Terms

Resource Management

Keywords

Science Cloud, High Throughput Computing, Job Profiling, Cloud Provisioning, PCA(Principal Components Analysis)

1. INTRODUCTION

Cloud computing nowadays enables numerous scientists to earn advantages by serving on-demand and elastic resources whenever they desire computing resources. This science cloud paradigm has been actively developed and investigated to satisfy requirements of the scientists such as performance, feasibility and so on. However, effective allocation and provisioning virtual machines on clouds are still considered as a challenging issue in scientists using high throughput computing, since it determines whether they can earn benefits from economy of scale in clouds or not. Moreover, allocating the "right" cloud resources(i.e., virtual machine type, cloud service site) on an optimal data center is very important as performance can vary widely depending on where and under what circumstances it actually runs. In these reasons, it is required that an appropriate and suitable model for science cloud to support increasing scientists and computations. Fortunately, job history which potentially involves status of resources where jobs are executed as well as properties of applications can lead us to devise performance-optimized provisioning scheme with meaningful factors through an appropriate processing of the traces.

In this paper, we present a ***PHAP***(Profiling Historical factor based Allocation and Pro- visioning model) which is

an allocation and provisioning model of science cloud, especially for high throughput computing applications. In this model, we utilize job traces where statistical method is applied to pick the most influential features for improving application performance. With the feature, the system determines where VM is deployed (allocation) and which instance type is proper (provisioning). An adaptive evaluation step which is subsequent to the job execution enables our model to adapt to dynamical computing environments. We show performance achievements as comparing the proposed model with other policies through experiments. Finally, we expect that improvement on performance as well as reduction of cost from resource consumption through our model.

The rest of this paper is structured as follows. Section 2 discusses related work and Section 3 presents introduction of the proposed system model where application model will be introduced. We discuss allocation and provisioning model in details in Section 4 , while Section 5 discusses experiment and its results. Finally, we conclude this paper and discuss future work in Section 6.

2. RELATED WORK

As growing attentions to clouds in several science communities, a lot of efforts to provide optimized and facilitated virtual environments on science clouds have been made. It is associated with that almost science applications typically involve long-term computer executions and huge dataset processing and large-scale computations requiring the availability of abundant computing infrastructures. Therefore, it is important to identify requirements of users as well as applications and to allocate adequate quantities of resources. Here, we are going to introduce several related works focused on two aspects; **1)** cloud provisioning and resource allocating issue **2)** utilization of job traces.

Lei Wang et al.[11] had investigated cloud modeling for scientific applications and evaluated their model using two kinds of workloads-MTC(Many Task Computing) and HTC (High Throughput Computing). The model adopted a policy which is based on the ratio of waiting jobs to total available VM to lease VM.

Another work, [12] proposed a controlling system which allocates appropriate resources through monitoring and anal -ysing current workloads of applications. In [12], a virtual server is operated on a group of physical machines and each server is responsible for particular application on the heterogeneous machines. The system, in addition, predicts future workload through analysing resource utility and real-time requirements of applications and schedules using the predicted information. In this way, diverse physical machines can be maintained in optimal status.

However, the above two studies mainly focused on the proper resource management and utilization without considering performance issue.

On the other hand, there also exist some researches using job histories. One of them [6] had proposed for predicting application runtime and queue wait time. In [6], it had exploited genetic algorithm to search similar jobs among job histories. This study predicts two metrics by mining historical workloads and the two are correspond to job execution time and wait time in queue. It firstly searches the most similar job among the job histories using genetic algorithm considering both characteristics of applications and resources. With the discovered one, it predicts two metrics (application runtime and queue wait time) using instance-based learning techniques. However, the main focus of the [6] do not lie on resource managements, but only runtime predictions.

Meanwhile, Bhuvan et al. [10] also exploited application profiling in shared resources to offer guaranteed performances as well. Their method for analysis, on the other hand, is different with ours since they consider only an aspect among various features.

An allocation and provisioning model we present focuses on the followings: First, selection of a cloud site which serves optimal performance without bursty workloads and results in efficient resource allocation. Secondly, a proper virtual machine provisioning which contributes to satisfy requirements of user and application. To determine the above two, our model utilizes three representative factors through profiling job traces. It is performed adaptably by evaluating profiled results and enables to adapt to current status of resources regardless of failure or overloading on systems.

3. SYSTEM MODEL

3.1 Target Application Model

Our model mainly focused on two kinds of workload types; High Throughput Computing (High Throughput Computing) and Many Task Computing (MTC) are generally found in most of workloads from various scientific applications. These kinds of workloads which are also called as 'Bag-of Tasks(BoT)' have several characteristics in common like the followings:

- Massively Paralleled
- Independent in each task (minor dependent tasks)
- Adoption of '*throughput*' as a main metric(over a fixed period of time)

Assume that a job(j_i) is submitted by some user has $\boldsymbol{n}$ number of tasks($t_{x+1}, t_{x+2}, \cdots, t_{x+n}$, here x is arbitrary job id) by parameter sweeping. The running time of each task is associated with overall total makespan of a job(j_i). '*throughput*' indicates the number of completed tasks in a fixed period of time and so we can induce a throughput of j_i as the following(Eq. 1).

$$throughput(j_i) = \frac{\boldsymbol{n}}{makespan(j_i)} \quad (1)$$

To increase throughput of j_i, we have to reduce makespan of j_i where all of n number of tasks should be completed since $\boldsymbol{n}$ is fixed by user when he submits the job. However, it is a hard work to predict makespan even though duration of one task is given, since each of them can have dynamic execution time by varied parameter value with independent operations. Hence, it is essentially required to recognize properties of the submitted job(i.e. tasks).

3.2 System Model

Figure 1 shows an abstract architecture where proposed **PHAP**(Profiling Historical factor based Allocation and Provisioning model) model is applied. It consists of five layers in total and main layer is the third layer named as '*Profiling &Provisioning Layer*'. Once users desire to compute their application, make their requests through the first layer denoted as '*User request Layer*'. The made requests in the first layer are sent to the next layer; '*Job Management*

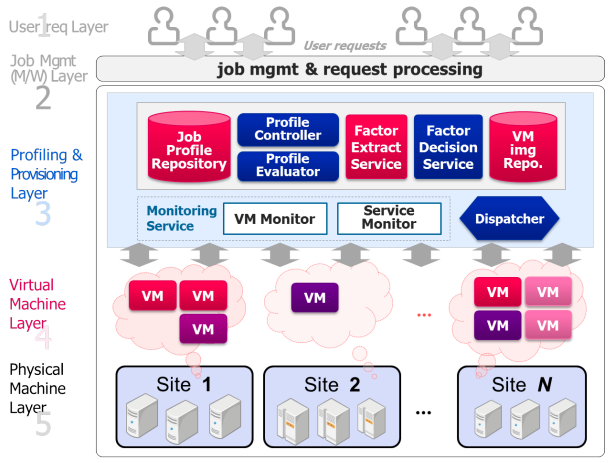

Figure 1: Abstraction of Overall System Model

Layer'. This layer is responsible to take requirements from the above layer and process them by producing job description files. According to the described information, a set of tasks for a job is produced and their information is recorded on database. After then, the third layer referred as *'Profiling &Provisioning Layer'* prepares job submissions through decisions of a cloud site to be submitted and an instance type using profiling services(*Factor Extract & Factor Decision Service*). The decisions are made by '*dispatcher*' and it periodically interacts with '*Factor decision service*' to get profiling information and 'Monitoring service' to check status of virtual machines as well as tasks on them. '*Monitoring Service*' collects status of virtual machines and tasks and records dynamic information of each cloud site. Job profile repository stores job profiles by mining some important information as a form of profile schema whenever all of tasks in each job are completed. Profiling evaluator and controller manage the stored profiles by evaluating with credit and controlling profiles with add/remove commands. *VM image repository* is a storage to keep VM backup images and its instances can be created using the stored snapshots. *Virtual Machine layer* which is located below the third layer represents virtual machine pool where VMs have diverse properties. The fifth layer named as '*Physical Machine Layer*' includes several cloud sites(cloud data centers), since we assume that there are multiple cloud sites(also denotes data center) which are located on geographically distributed placements. Each cloud site is connected to the other cloud sites with WAN(Wide-Area-Network) and users can access to them only through a cloud service provider. Each site updates their monitoring information like total number of jobs, the number of failed jobs, etc. to central DB server by *Monitoring service*. In a practical environment, it is possible to apply the existing integrated middleware framework over the second and third layer and to operate the overall allocating and provisioning cycle in accordance with the cycle of middleware. In fact, we are processing to implement this model on an existing framework, *HTCaaS*[9] which is adopting an agent-based multi-level scheduling mechanism. Hence, some modules such as *Dispatcher*, *Monitoring service* are able to be integrated or replaced into modules with same roles in HTCaaS. For example, Agent manager and Job manager which are components of HTCaaS can cover deploying VM instances and submitting jobs respectively instead of Dispatcher. Thus, when an event for job submission is occurred, a virtual machine having agent (an agent requires only one core) is deployed to the chosen site and the agent pulls a task. The details of job submission will be explained on the next section together with allocation & provisioning algorithms.

4. ALLOCATION AND PROVISIONING MODEL

4.1 A Processing Cycle

Overall procedure of PHAP is classified into the three great divisions; **(1)**Profiling, **(2)** Allocation & Provisioning, **(3)**Evaluating.

The first division is to derive principal information through analysing job traces. Before performing the analysis, two requisite conditions should be accomplished. One is checking whether enough profiles exist and the other is mining information from collected traces in a scheme defined. After that, decisions for allocating and provisioning are made using principal factors extracted. Once all executions for a request(job) is completed, evaluating is performed by calculating a difference between referred profile and new records of job just completed. The details of above three division will be explained on the following subsections in order of sequence.

4.2 Profile Analysis

As mentioned previously, our model exploits kernel information by refining job profiles. Exploiting job profiles has a lot of advantages in diverse aspects such as estimating or predicting job runtime, measuring performance of computing resources and exploring system errors, etc. Above all, it potentially involves status of computing resources where jobs run as well as application properties which are essentially needed for execution. However, these traces typically exist in a large-scale and they are required to be processed and refined properly so that picking out useful features for the performance improvement. This section discusses how to perform the job profiling and apply the resulted kernel data to Clouds through a practical example.

To analysis and explore features from tasks, we employed PCA(Principal Component Analysis)[4] which is one of the well-known technique from statistics for simplifying large-scale multivariate data. As sorting dimensions in order of importance, we can extract important information which is called *Principal Components*(PC) and can discard low significance dimensions with minimal effort. Particularly, we perform the technique to pick the most powerful one among factors(parameters) describing a task. Before applying the analysis, we need to define factors which are able to describe a task as well as have much relations with performance.

Factors to be determined are categorized according to two aspects; *Application* and *Resource*. As application factors, several arguments or inputs used during runtime can be considered, since they have high possibilities to affect execution time of the applications directly. Whereas a resource aspect, physical parameters which have a large impact on duration of jobs are included such as CPU and network related factors(e.g., *the number of cores, CPU power, network bandwidth, computing resource location[distance]* etc).

Table 1: An Example of Determined Factors

-	Factor name	Category	Property
f_1	*LIGANDS id*	Application Factor	-
f_2	*PROTEIN id*		-
f_3	*ga_num_eval*		-
f_4	*ga_run*		-
f_5	*ga_pop*		-
f_6	*ga_generation*		-
f_7	*num_run*		-
f_8	*Distance*	Resource Factor	*static*
f_9	*Reliability*		*dynamic*
f_{10}	*# of CPU*		*static*
f_{11}	*Network cost*		*static*
f_{12}	*Avg.Waiting time*		*dynamic*
f_{13}	*Agent Reliability*		*dynamic*
f_{14}	*Execution Time*		

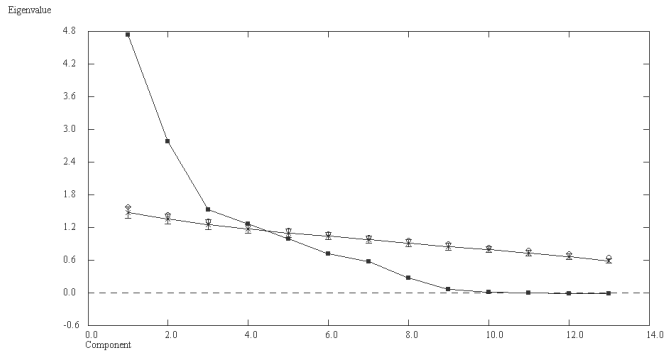

Figure 2: A Scree Plot of Analysis

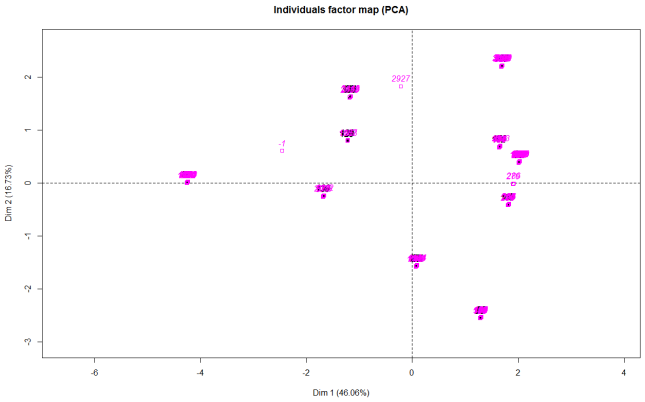

Figure 3: Plots of PCA scores

$$\mathbf{PC} =< f_{prin\_a}, f_{prin\_r}, \mathbb{J}_{prin}, credit > \quad (2)$$

$$\boldsymbol{M} = \begin{bmatrix} t_{1,1} & \cdots & t_{1,d} \\ \vdots & \ddots & \vdots \\ t_{n,1} & \cdots & t_{n,d} \end{bmatrix} \quad (3)$$

$$EigenVector_5 = (.017, -.011, .073, \cdots, .412, .205) \quad (4)$$

$$\begin{aligned} PCscore\_t_1 &= 0.017 \times t_{1,1} + \cdots + 0.205 \times t_{1,13} \quad (5) \\ &= 1.634 \end{aligned}$$

Table 2: An Example Monitoring Information of Cloud Sites in Amazon EC2

Name	f_8	f_9	f_{10}	f_{11}	f_{12}	f_{13}
Virginia	1	0.61	8	5	2	0.45
Oregon	-1	0.20	3	5	-1	0.91
California	-1	1.00	6	5	8	0.75
Ireland	2	0.91	7	4	5	1.00
Singapore	3	1.00	4	2	3	1.00
Sydney	3	0.99	3	3	6	0.98
Tokyo	4	0.96	5	-1	7	0.99
Sao-paulo	1	0.61	-1	5	4	0.86

However, it depends on information which are monitored on each cloud site.

Table 1 shows an example about how to define factors of a practical application that we employed as an actual application, Autodock[8]. In this example, we handle seven application factors which are adopted as arguments or input properties ($f_1, \cdots, f_7$) and f_3 - f_7 among them are treated as common variables for parameter sweeping. In case of resource, a set of elements like ($f_8, \cdots, f_{13}$) are extracted from monitoring information of each cloud site. These factors are classified into two types; static or dynamic. Static factors typically refer to unchangeable properties since the site is registered as available data center. "*Distance*, *# of CPU and Network cost* " belong to that type. On the other hands, there exist dynamically changed arguments such as "*Reliability*, *Avg.Waiting time* and *Reliability of Agents*" since they are depending on the current status of jobs on each site. *Reliability* is a rate of the number of successful one over total tasks and the number changes continually. In this example, we supplemented resources factors; *Agent Reliability* which means the rate of the number of successful things over total submitted agents and *Agent Waiting Time*, as well. The corresponding values for each factors are determined in different ways each other. In case of application factors, the values themselves which are specified by user at the submission are used.

For resource factors, on the other hands, we adopt rank score which derives from ranking them with respect to proportion relation with execution time and accords values in ascending order of rank. In other words, a site having the topmost rank will score a high grade. The element which has the lowest rank becomes to get '-1' instead of '0'. In case of *Distance*, each site gets score with the number of hops from submission site, and then converts the score with respect to rank between them in an identical way of the previous method. Table 2 shows an expression example of resource factors for cloud sites in Amazon EC2 [2] where f_8 and f_{10}-f_{12} are expressed with the rank score as mentioned previously.

Overall processing of the profiling is carried out as follows:

1) *Construction of the Input Matrix:* firstly, construct (n x d) input Matrix $\boldsymbol{M}$ (Eq. 3) with $\boldsymbol{n}$ number of profiles where each row is a task vector(t_{id}) having $\boldsymbol{d}$ number of factors(as notated on Table 6).

2) *Quantile Normalization:* secondly, apply the quantile normalization to $\boldsymbol{M}$. It intends to convert the data in a variety of ranges into them having an identical range each other based on quantile rank.

Table 3: Example profiles of Autodock application

id	*app* factors of t_{id}							r	s	vm_{id}	$T_{exec}(id)$
	f_1	f_2	f_3	f_4	f_5	f_6	f_7				
276	1	1	300000	50	150	270000	50	Sydney	*Done*	*m3.xlarge*	150
277	2	1	300000	50	150	270000	50	Oregon	*Failed*	*m1.small*	126
...	...							...			
572	3	1	600000	50	150	27000	100	Tokyo	*Done*	*t1.micro*	290
573	4	1	600000	50	150	27000	100	Tokyo	*Done*	*m1.large*	232

Table 4: converted profiles of the above example

id	*app* factors of t_{id}							*res* factors of t_{id}						s	vm_{id}	$T_{exec}(id)$
	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}	f_{11}	f_{12}	f_{13}			
276	1	1	300000	50	150	270000	50	3	0.99	3	3	6	0.98	*Done*	*t1.micro*	150
277	2	1	300000	50	150	270000	50	−1	0.20	3	5	−1	0.91	*Failed*	*m1.small*	126
...	...							...						...		
573	4	1	600000	50	150	27000	100	4	0.96	5	−1	7	0.99	*Done*	*m1.large*	232

Table 5: Applying PCA to profiles

id	*app* factors of t_{id}							*res* factors of t_{id}						$T_{exec}(id)$
	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}	f_{11}	f_{12}	f_{13}	
276	–	6.6	2.5	6.6	6.7	6.7	6.6	6.6	2.5	2.5	6.7	6.6	6.69	6.6
277	–	6.7	2.5	6.6	6.7	6.7	6.6	6.6	2.5	2.5	6.7	6.6	6.71	6.6
...	... ...													...
572	–	6.7	6.7	6.7	6.7	2.52	6.71	6.71	6.7	6.7	6.7	6.1	6.7	6.7
573	–	6.7	6.7	6.7	6.7	2.52	6.71	6.71	6.7	6.7	6.7	6.1	6.7	6.7
Eigen Value	-	-9.55E -16	0.58	2.79	4.74	1.27	0.02	0.997	1.54	0.28	0.067	4.32E -15	-2.67E -15	0.72
Rank	-	*6*	*4*	*2*	**1**	*3*	*5*	*2*	**1**	*4*	*5*	*6*	*7*	*3*

3) Principal Components Analysis: carry out PCA by calculating correlation matrix $\boldsymbol{C}$ of input matrix $\boldsymbol{M}$. Subsequently, calculate *eigenvalues* and *eigenvector* sets which exist as much as the number of factors.

4) Election of 1st PC and Calculation of PCA score: the *eigenvalue* which is the highest one becomes the 1st PC and PCA scores are calculated using eigenvector which is corresponded with eigenvalue of 1st PC. Among the results, profile *id* having the highest PCA score become a representative job profile.

Here, we make use of '*eigenvalues*' and '*eigenvectors*' which are the variances of the objects on each PC and the rotation vectors from the PC space back to the landmark shape, respectively. Thus, calculating [*eigenvectors* * normalized_scores+ *consensus*] gives us a shape model from a particular point in the PC shape space.

As time advanced, profiles become to be accumulated more and more in a determined scheme which is denoted on Eq. 2. The accumulated profiles for the above example can be described as Table 3. After then, the resource factors are determined depending on a monitoring information (Table 2) and the converted profiles are described as Table 4. A *credit* value is decisioned initially as -1 and it will be controlled during '*Evaluation step*'.

When the analysis is achieved, the values in profiles are converted to a normalized form by quantile normalization and the results can be shown on Table 5 and Figure 2 as scree plot graph. As a result, we can notice that application and resource factor having the highest eigenvalue among each category is f_5 and f_9, respectively. Accordingly, the result leads to that principal factor for application(f_{prin_a}) is f_5 and resource's one(f_{prin_r}) is f_9. Moreover, an eigenvector matched with the highest eigenvalue(f_5, here, f_5's one is higher than f_9's one) is $EigenVector_5$ as depicted on Eq. 4. Using the vector, we can compute $PCscore$ for all tasks and a calculation example of it is depicted on Eq. 6. After applying Eq. 6 to all tasks, PCscores for all the tasks are calculated as shown on Figure 3 and a set of the tasks which have the highest score are included to $\mathbb{J}_{prin}$ in Eq. 2.

Table 6: Notations for PHAP model

-	Property	Description
P_{id}	[t_{id}, vm_{id}, r, $T_{exec}(id)$, s]	a profile tuple
t_{id}	$<f_1, f_2, \cdots, f_d>$	task vector includ -ing t_{id}'s parameters
vm_{id}	$vm_{id} \in$ { *Instances* }	*vm* instance type
r	$r \in$ { *us-east,* $\cdots$ *etc.* }	cloud site name
s	$s \in$ { *done, failed cancelled, waiting*}	status of a task

4.3 Algorithms of PHAP Model

Once a job is submitted, it becomes to split into multiple tasks having various arguments. To allocate resource and provision virtual machine, first of all, check an existence of PC as shown on Algorithm 1(line 3). If there is no PC, carry out **PCA** with a list of the determined factors and recent job traces. However, if there are no traces yet, we select a cloud site according to Round-Robin order until a system get enough profiles. If we gain PC through the analysis, **ProfileSelect** function is achieved(line 6) with the three results; f_{prin_a}, f_{prin_r} and each task vector as requested. In

Algorithm 1 Cloud Allocation and Provisioning Algorithm based on Profile Analysis

1: PC = *null* , t_{new} = *null*, $PC_{selected}$ = *null*
2: F={*defined factors*}, $\mathbf{t}_{recent}$ is a set of w recent profiles.
3: **if** PC not exists **then**
4: PC = **PCA** (F, $\mathbf{t}_{recent}$);
5: **end if**
6: $P_{selected}$ = **ProfileSelect**($PC.f_{prin\_a}$, $PC.f_{prin\_r}$, t_{new})
7: $r_{selected} = P_{selected}.r$
8: vm_{new}=$vm(PC.\mathbb{J}_{prin} \cap P_{selected})$
9: **if** vm_{new} on $r_{selected}$ not exists **then**
10: deploy vm_{new} on $r_{selected}$
11: **end if**
12: // {vm_{new} is the one having same type with a decisioned vm type for new task. }
13: *Schedule* t_{new} on the *Queue*
14: **Evaluate** ($T_{exec}(new)$, $T_{exec}(selected)$);

Algorithm 2 ***Profile Select*** Algorithm

Input: f_{prin_a}, f_{prin_r}, t_{new}
1: $\mathbf{P}_{candidate}$ = *null* , $\mathbf{P}_{selected}$ = *null*
2: $\boldsymbol{F}$ = $\{f_1, ..., f_a, .., f_d\}and\{1 \leq a \leq d\}$, $f_{prin\_a} \in \{f_1, .., f_a\}$, $f_{prin\_r} \in \{f_{a+1}, .., f_d\}$
3: $\mathbf{P} \in JobHistory$
4: **repeat**
5: each $\mathbf{P}$ In *JobHistory*
6: Find $\mathbf{P}$ where P.$j(f_{any|1\leq any\leq a})$==$t_{new}(f_{prin\_a})$
7: $P_{candidate} = P_{candidate} \cup \mathbf{P}$
8: **until** $\mathbf{P}_{uncheck}$ == *empty*
9: $P_{selected} = MIN(P.T_{exec})$ && $MAX(P.r(f_{prin\_r}))$
Output: $P_{selected}$

order to choose appropriate profile to be used for allocation and provisioning, we follow Algorithm 2. For all of existing profiles, search profiles whose application factor has the same value with that of requested task and let the profile include in a set of candidate profiles(line 8). After then, elect profiles which has maximum score of resource factor and minimum execution time(line 9). Finally, it returns the satisfied profile(s) as $P_{selected}$. Therefore, it provides a site information as $r_{selected}$(line 7 of Algorithm 1) and decides a type of vm based on intersection results between representative task set($\mathbb{J}_{prin}$) and the chosen profiles. Once a type of vm is decided, it examines the cloud site($r_{selected}$) to check whether a vm which is the same type with vm_{new} is available, or not. If available, it schedules the task into the vm, otherwise, it deploys vm_{new} to $r_{selected}$. When a task is finished, evaluate it by calculating difference between its execution time and the profile we referred to. The difference is reflected on credit for PC and if the credit is under -2, it performs **PCA**(line 4 of Algorithm 1) again with recent profiles. The details about adaptive evaluation is indicated on the previous work[5].

5. EXPERIMENTS

5.1 Experimental Setting

We performed this experiment based on measurements over real systems which provides management services of HTC jobs and IaaS services, referring to HTCaaS[9]. HTCaaS system offers *Agent-based* job execution service and we had to adjust our scenario in accordance with its *agent-based* concept so that constructing a proper experiment environment. For cloud infrastructure, in addition, we adopt Amazon EC2 which is one of the well-known commercial services of public cloud. Accordingly, we configured a network topology for the experiment based on data centers in Amazon EC2. Figure 4 depicts the configured topology. As described on that, suppose that a service provider is located on Korea(Daejeon) and that each site is connected via WAN. In addition, applying and implementing of PCA to HTCaaS is based on Flanagan Scientific Library[7]. We ran a real application which is Autodock as previously addressed and basically used 10 targets for docking.

Figure 4: Network topology of Amazon EC2 Site

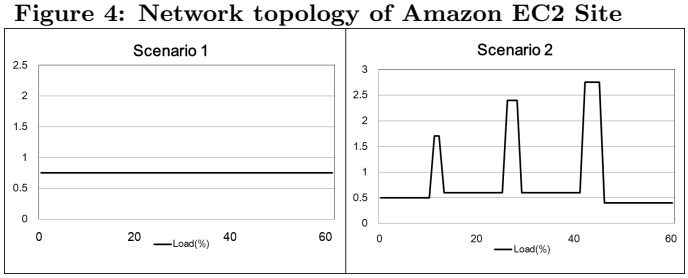

Figure 5: Arrival Pattern of Scenario 1, 2

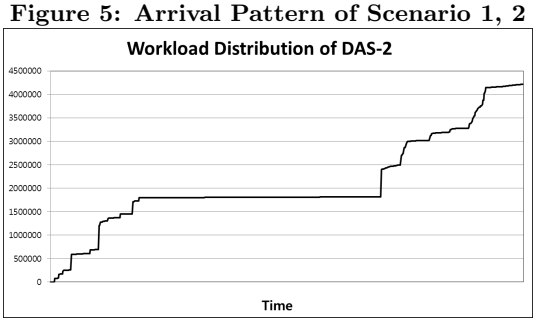

Figure 6: Arrival Pattern of DAS-2

5.2 Scenarios

For performance evaluation, we ran the experiment with several scenarios and compared our model to three models on each scenario. We performed the experiment in accordance with three types of workloads and properties as the following scenarios:

- ***Scenario 1*** uniformed workload / randomly generated parameter combinations
- ***Scenario 2*** synthetic workload consisting of uniform and bursty patterns(Figure 5)/ randomly generated parameter combinations
- ***Scenario 3*** workload following DAS-2[3]'s arrival patterns(Figure 6) / randomly generated parameter combinations(it matches to each task randomly)

The first scenario is to show improvement of performance on environment with simple patterns of workload without any interferences and the second one is to evaluate the performance in an environment with unexpected and sudden overloads. On the third scenario, in case of DAS-2 [3] workloads which is from GWA(Grid Workload Arhieve)[1], its arrival pattern shows increasing tendency throughout entire time. All patterns of the above scenarios are depicted on Figure 7, 8 and 9. For performance metrics to be used for comparisons, there are *throughput*, *makespan time*, $WaitingTime$. As policies to compare with our model, we employed Random, Round-Robin and Best-Effort selection methods. In case of Best-Effort, it intends to select a site first having the minimal tasks among entire sites. Overall, we ran one thousand(1000) number of jobs for five times and exploited the average of results for performance evaluation. Additionally, we performed a supplementary experiment to explore appropriate size of profile to be used for profiling.

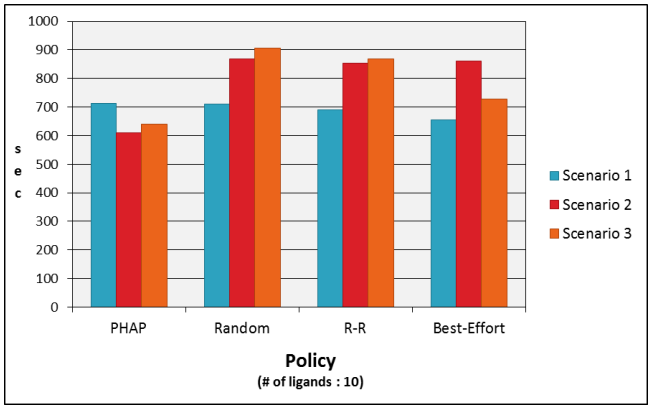

Figure 7: Experimental Results(makespan)

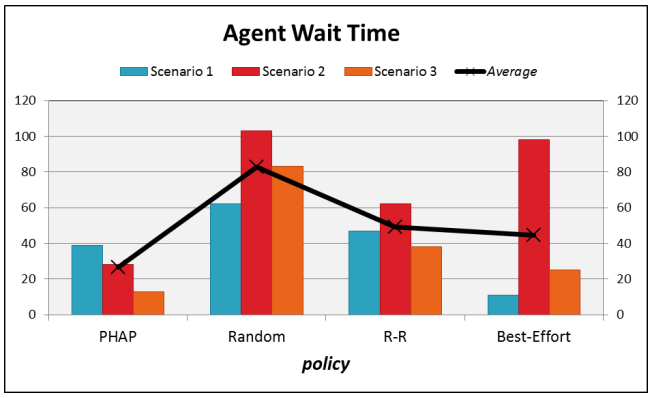

Figure 8: Avg. Agent Wait Time

5.3 Results

With respect to the addressed conditions, results of the evaluation are as follows. Figure 7, 8 depicts the comparison results in terms of makespan time and waiting time, respectively.

According to the result of scenario 1 on the Figure 7, overall results have similar durations, but makespan time of the Best-Effort is resulted in the least time and PHAP leads to

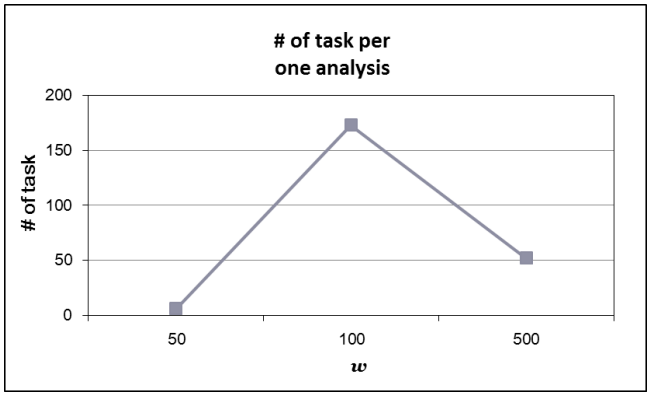

Figure 9: The number of tasks per one PCA according to w size(makespan)

the highest makespan time. In case of Best-Effort policy, the result shows a balanced workload distribution for each cloud site, since it chose a site having the minimal jobs and it leads to other lower waiting time than the three of policies including PHAP. In addition, it is easy to intend a good result since there are no sudden overloads during the overall execution. However, the maximum difference between Best-Effort and PHAP is measured as approximately 41 seconds and this difference seems a negligible amount as compared to overall duration. In the results of scenario 2, our PHAP model leads to the least makespan time and other all of three conditions shows the longer makespan time than the previous scenario. In this scenario, the maximum difference between them is over 250 second and our model achieved better performance. It seemed that our proposed model is possible to choose best resource based on historical records and to do so adaptably. On scenario 3 which is based on the more realistic arrival pattern than other two scenarios, moreover, the PHAP model also took the minimum time among them. It seemed that the proposed model leads to an accurate selection of site on sudden overloaded environment. The Figure 8 shows the another performance results. As we utilized HTCaaS system which uses *Agent* concept, we measured the wait time of Agent. It also implies how much the model can deduce the decision of an allocated site correctly. In this results, Best-Effort policy has the shortest wait time on the first scenario, since it chooses a site having the least running tasks, that is, it adopts the one having the shortest waiting time.

Therefore, it seems relatively can gain the shortest waiting time. On the other hand, overall average of waiting time is the shortest on PHAP model, since it is able to endure unexpected overload or increasing loads like scenario 2 and 3. Therefore, our PHAP model can contribute to overall performances by reducing waiting time as well as optimizing makespan time which is able to lead to better throughputs.

In addition to these, we ran a supplementary experiment in which w size is controlled. w refers to the number of recent profiles that we will use in analysis. By carrying out the experiment, the optimal range of (window) size in profiles can be grasped. The result shows that profile size to be used in analysis is the most of appropriate on the size is approximately 100, where the system affords to take more tasks in an analysis.

6. CONCLUSION AND FUTURE WORK

This paper proposed PHAP model which is an adaptive cloud model for allocation and provisioning using historical factor analysis. The model can effectively construct a scientific cloud infrastructure for HTC. To analysis the profiles of jobs, we applied PCA technique as a statistical method to elect the effective factors. The factors are employed for selecting cloud site and deciding proper virtual machine type.

The performance evaluation is performed on HTCaaS system which is the agent-based multi-level scheduling system and its results shows our model can improve overall performance of HTC applications as compared with other policies such as Random, Round-Robin and Best-Effort.

In the near future, we will implement our PHAP model in the HTCaaS completely and report performance analysis based on diverse computing infrastructures and scheduling policies.

7. REFERENCES

[1] G. W. Archive(GWA). http://gwa.ewi.tudelft.nl/.

[2] A. E. C. Cloud). Available at http://aws.amazon.com/ec2.

[3] DAS2-Grid. Available at http://cs.vu.nl/das2.

[4] I. T. Jolliffe. *Principal Component Analysis*. Springer, second edition, Oct. 2002.

[5] S. Kim and Y. Kim. Application-specific cloud provisioning model using job profiles analysis. In *High Performance Computing and Communication 2012 IEEE 9th International Conference on Embedded Software and Systems (HPCC-ICESS), 2012 IEEE 14th International Conference on*, pages 360–366, 2012.

[6] H. Li, D. Groep, and L. Wolters. Efficient response time predictions by exploiting application and resource state similarities. In *Grid Computing, 2005. The 6th IEEE/ACM International Workshop on*, pages 8 pp.–, 2005.

[7] F. S. Library. http://www.ee.ucl.ac.uk/ mflanaga/java/.

[8] G. M. Morris, D. S. Goodsell, R. S. Halliday, R. Huey, W. E. Hart, R. K. Belew, and A. J. Olson. Automated docking using a lamarckian genetic algorithm and an empirical binding free energy function. *Journal of Computational Chemistry*, 19(14):1639–1662, 1998.

[9] S. Rho, S. Kim, S. Kim, S. Kim, J.-S. Kim, and S. Hwang. HTCaaS: A Large-Scale High-Throughput Computing by Leveraging Grids, Supercomputers and Cloud. In *Research Poster at IEEE/ACM International Conference for High Performance Computing, Networking, Storage and Analysis (SC12)*, Nov. 2012.

[10] B. Urgaonkar, P. Shenoy, and T. Roscoe. Resource overbooking and application profiling in a shared internet hosting platform. *ACM Trans. Internet Technol.*, 9(1):1:1–1:45, Feb. 2009.

[11] L. Wang, J. Zhan, W. Shi, and Y. Liang. In cloud, can scientific communities benefit from the economies of scale? *Parallel and Distributed Systems, IEEE Transactions on*, 23(2):296–303, 2012.

[12] X. Wang, D. Lan, G. Wang, X. Fang, M. Ye, Y. Chen, and Q. Wang. Appliance-based autonomic provisioning framework for virtualized outsourcing data center. In *Autonomic Computing, 2007. ICAC '07. Fourth International Conference on*, pages 29–29, 2007.