Cluster Comput
DOI 10.1007/s10586-014-0371-2

Towards effective science cloud provisioning for a large-scale high-throughput computing

Seoyoung Kim · Jik-Soo Kim · Soonwook Hwang · Yoonhee Kim

Received: 29 September 2013 / Revised: 25 February 2014 / Accepted: 15 March 2014

Abstract The science cloud paradigm has been actively developed and investigated, but still requires a suitable model for science cloud system in order to support increasing scientific computation needs with high performance. This paper presents an effective provisioning model of science cloud, particularly for large-scale high throughput computing applications. In this model, we utilize job traces where a statistical method is applied to pick the most influential features to improve application performance. With these features, a 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 by comparing the proposed model with other policies through experiments and expect noticeable improvements on performance as well as reduction of cost from resource consumption through our model.

Keywords Science cloud · High-throughput computing · Job profiling · Cloud provisioning · PCA (Principal components analysis)

S. Kim · J.-S. Kim · S. Hwang
National Institute of Supercomputing and Networking, KISTI, Daejeon 305-806, Korea
e-mail: sssyyy77@kisti.re.kr

J.-S. Kim
e-mail: jiksoo.kim@kisti.re.kr

S. Hwang
e-mail: hwang@kisti.re.kr

Y. Kim (✉)
Department of Computer Science, Sookmyung Women's University, Seoul 140-742, Korea
e-mail: yulan@sookmyung.ac.kr

1 Introduction

Cloud computing nowadays enables numerous scientists to earn advantages by serving on-demand and elastic resources whenever they desire 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, allocating and provisioning virtual machines effectively are still considered as a challenging issue for 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. For these reasons, an appropriate and suitable model for science cloud which can support increasing scientists and computations is required. Fortunately, job profiles which potentially involve 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 Profiling Historical factor for Allocation and Provisioning on Science cloud (**PHAP**) model which is an allocation and provisioning model of science cloud, especially for high throughput computing applications as well as many task computing. In this model, we utilize job traces where a statistical method is applied to pick the most influential features for improving application performance. With the features, 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 achieve-

Published online: 19 April 2014

ments by comparing the proposed model with other policies through experiments. Finally, we expect improvement on performance as well as reduction of cost from resource consumption through our model.

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

2 Related work

With increasing attentions to clouds in several science communities, a lot of efforts have been made to provide optimized and facilitated virtual environments on science clouds. Moreover, almost science applications typically involve long-term computer executions, huge dataset processing and large-scale computations requiring the availability of abundant computing infrastructures. Therefore, it is important to identify requirements of users and their 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.

Wang et al. [1] 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 in order to lease VM. Another work, [2] proposed a controlling system which allocates appropriate resources through monitoring and analysing current workloads of applications. In [2], a virtual server is operated on a group of physical machines and each server is responsible for particular application on the heterogeneous machines. In addition, the system predicts future workload through analysing resource utility and real-time requirements of applications and schedules virtual machines 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, [3] had proposed predicting application runtime and queue wait time. In [3], genetic algorithm had exploited to search similar jobs among job histories. This study predicts two metrics by mining historical workloads corresponding to job execution time and wait time in queue. It firstly searches the most similar job among the histories based on genetic algorithm considering characteristics of both 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 [3] do not lie on resource managements, but only runtime predictions.

Meanwhile, Bhuvan et al. [4] 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 both 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 allows 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 focuses on two kinds of application types; High throughput computing (HTC) and many task computing (MTC) [5] are generally found in most of workloads from various scientific applications. These kinds of workloads which are also referred as 'Bag-of Tasks (BoT)' have several characteristics in common as the followings:

- Massively Paralleled
- Loosely coupled or uncouple in each task
- Adoption of '*throughput*' as a main metric (over a fixed period of time)

Assume that a job(j_i) is submitted by some user and has $\boldsymbol{n}$ number of tasks($t_{x+1}, t_{x+2}, \ldots, t_{x+n}$, here x is arbitrary job id) using 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).

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

To increase throughput of j_i, we have to reduce its makespan where every 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 varying parameter value with independent operations. Hence, it is essentially required to recognize properties of the submitted job (i.e. tasks).

Here we adopt two representative applications as an example of such a case to verify the superiority on such workloads and to explain about how to support them. The applications are Autodock [6] and Madgraph5 [7].

Autodock is a suite of automated docking tools used in Pharmaceutical research domain for new drug discovery by simulating protein docking.

Madgraph5 is a matrix element generator (simulation tool) in High Energy Physics for simulating QCD (Quantum chromodynamics) strong interation of fundamental particles at hadron colliders. This application is based on Monte-Carlo simulation which is considered as 'embarrassingly parallel' and is split into smaller uncoupled sub-jobs with no communications on distributed environments.

All of these applications usually have a large number of computations (a target protein of 3CL-pro of SARS with 1.1 million chemical compounds for protein docking and 5 million particle collisions resulting in tens of millions of interactions for QCD simulations) and require substantial amount of computing time.

In the following section after system model, we will explain full-cycle with examples of these practical applications.

3.2 System model

Figure 1 shows an abstract architecture where proposed **PHAP** method is applied. It consists of five layers in total and the third one ('*Profiling and Provisioning Layer*') is the layer where the proposed method is actually implemented.

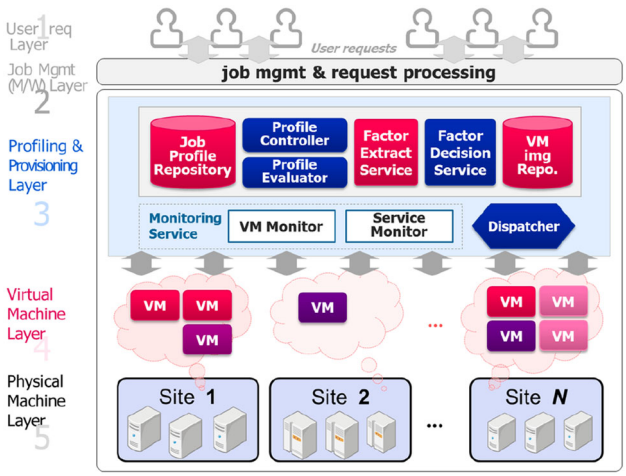


Fig. 1 Abstraction of overall system model

Once users desire to compute their application, the system takes 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 Layer'*. This layer is responsible for taking requirements from the above layer and process them by producing job description files. In a practical environment, it is possible to apply the existing integrated middleware framework (such as HTCaaS) over the second layer and to operate the proposed allocating and provisioning cycle in accordance with the cycle of middleware.

In accordance with the described information, a set of tasks for a job is produced and their information are recorded on database. Then the third layer referred as *'Profiling and Provisioning Layer'* prepares job submissions through decisions of a cloud site to be submitted and an instance type using profiling services (*Factor Extract and 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 running 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 every 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 by 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 '*Physical Machine Layer*' includes several cloud sites, since we assume that there are multiple cloud sites (also denotes data center) which are located on geographically distributed area. Each cloud site is connected to the other ones with WAN (Wide-Area-Network) and users can access them only through a cloud service provider. Each site updates their monitoring information such as total number of jobs, number of failed jobs, etc. to central DB server using *Monitoring service*.

In fact, we are processing to implement this model on an existing framework, *HTCaaS* [8] which is adopting an agent-based multi-level scheduling mechanism and it can be applied on the second layer as a job management framework. 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 handle respectively deployment of VM instances and submission of jobs 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 this agent pulls a task.

The details of job submission will be explained on the next section together with allocation and provisioning algorithms.

4 Allocation and provisioning model

4.1 A processing cycle

Overall procedure of *PHAP* is classified into the three main categories; (1) Profiling, (2) Allocating and Provisioning, (3) Evaluating.

The first phase derives principal information by 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 as defined. After that, in the second step, decisions for allocating and provisioning are made using principal factors extracted. Once all executions for a request (job) are completed, evaluation is performed by calculating a difference between referred profile and new records of job just completed in the third step. Here, moreover, it is assumed that budget and hardware conditions for resources are not considered. It is only focused on overall performance. The details of above three steps will be presented on the next subsections in order of sequence.

4.1.1 Profiling

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. Meanwhile, these traces typically exist in a large-scale and they requires to be processed and refined properly in order to pick out useful features. This section discusses how to perform the job profiling and apply the resulted kernel data to Clouds through practical examples above mentioned.

To analyse and explore features from tasks, we employ PCA (Principal Component Analysis) [9] which is one of the well-known statistics technique for simplifying large-scale multivariate data. As sorting dimensions in order of importance, we can extract important information which are called *Principal Components* (PC) and can discard low significance dimensions with minimal effort. Particularly, we use the technique to pick the most powerful one among the factors describing a task. Before applying the analysis, we need to define factors which are able to describe a task and which have strongly related to 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 directly the execution time of the applications. The determined application factors must be able to distinguish sub-jobs (tasks). For a resource aspect, physical features 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). It typically depends on information which are monitored on each cloud site the system targets. Table 1 and 2 show the defined factors for the above mentioned practical applications.

Table 1 Factor decision for autodock, Ex1

-	Factor name	Category	Property
fa_1	*LIGANDS id*		-
fa_2	*PROTEIN id*		-
fa_3	*ga_num_eval*	Application	-
fa_4	*ga_run*	factor	-
fa_5	*ga_pop*		-
fa_6	*ga_generation*		-
fa_7	*num_run*		-
fr_1	*Distance*		*Static*
fr_2	*Reliability*		*Dynamic*
fr_3	*# cpu*	Resource	*Static*
fr_4	*Network cost*	factor	*Static*
fr_5	*Avg.waiting time*		*Dynamic*
fr_6	*Agent reliability*		*Dynamic*
fr_7	*Execution time*		

Table 2 Factor decision for madgraph5, Ex2

-	**Factor name**	**Category**	**Property**
fa_1	$f1$		`Scale`
fa_2	$f2$	Application	`Param`
fa_3	*ebeam*	factor	-
fa_4	*tot#_events*		-
fr_1	*Distance*		*Static*
fr_2	*Reliability*		*Dynamic*
fr_3	*# cpu*	Resource	*Static*
fr_4	*Network cost*	factor	*Static*
fr_5	*Avg.waiting time*		*Dynamic*
fr_6	*Agent reliability*		*Dynamic*
fr_7	*Execution time*		

In the first example (Ex1, Table 1), which is for auto-dock application, we handle seven application-factors which are adopted as arguments or input properties ($fa_1, \ldots, fa_7$). Among them, fa_3 to fa_7 are treated as common variables for parameter sweeping.

In the second example (Ex2, Table 2), which is for madgraph5, there are four application-factors (fa_1, fa_2, fa_3 and fa_4) for sweep values.

In case of resource for both examples, a set of elements ($fr_1, \ldots, fr_6$) are extracted from monitoring information of each cloud site. These factors are classified into two types; $static$ or $dynamic$. Static factors typically correspond to unchangeable properties since the site is registered as available data center. "*distance*, *# of cpu* and *network cost*" belong to such a type. On the other hands, there exist arguments such as "$reliability$, $avg.waiting\ time$ and *agent reliability*" that dynamically change, since they are depending on the current status of jobs on each cloud site as time passed. Here $reliability$ is a rate of the number of successful tasks over all, and *distance* is the number of hops from submission site to the chosen site.

In addition to common factors as previously mentioned, we supplemented with two additional factors: *agent reliability* which means the rate of the number of successful things over total submitted agents, and (*agent*) *waiting time* which is waiting time from submission of a job to beginning of the first task. Both metric are measured by HTCaaS. Each value (score) of the determined factors is determined in different ways. In case of application factors, it uses the values which are specified by user at submission.

For resource factors, on the other hands, we adopt rank score which derives from ranking them respecting proportion relation with execution time and arranging 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 initial score according to the number of hops from submission site, and then converts the score with respect to rank between them.

Table 3 An example monitoring information of cloud sites in Amazon EC2

Name	fr_1	fr_2	fr_3	fr_4	fr_5	fr_6
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

Table 3 shows an expression example of resource factors for cloud sites in Amazon EC2 [10] where *distance* and #*cpu*, *network cost* and *avg. waiting time*(2nd, 4–6th columns) are expressed with the rank score as mentioned previously.

Overall processes of the profiling is carried out as follows:

(1) *Construction of the Input Matrix:* firstly, construct (n x d) input Matrix $\boldsymbol{M}$ (Eq. 2) 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 7).

$$\mathbf{M} = \begin{bmatrix} t_{1,1} & \cdots & t_{1,d} \\ \vdots & \ddots & \vdots \\ t_{n,1} & \cdots & t_{n,d} \end{bmatrix} \tag{2}$$

(2) *Quantile Normalization:* secondly, apply the quantile normalization to $\boldsymbol{M}$. It intends to make identical range of data using quantile ranking.

(3) *Principal Components Analysis:* carry out PCA by calculating a correlation matrix $\boldsymbol{C}$ of input matrix $\mathbf{M}$. 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 highest *eigenvalue* becomes the 1st PC and PCA scores are calculated using eigenvector corresponding with eigenvalue of 1st PC. Among the results, a profile having the highest PCA score becomes a representative job profile.

Here, we make use of '$eigenvalues$' and '$eigenvectors$' which are respectively variances of the objects on each PC and the rotation vectors from the PC space back to the landmark shape. Thus, calculating the following equation,

$$[eigenvectors * normalized\_scores + consensus]$$

gives us a shape model from a particular point in the PC shape space. With time, profiles accumulate more and more in a determined scheme.

$$\mathbf{PC} =< fa_{prin}, fr_{prin}, \mathbb{J}_{prin}, credit > \tag{3}$$

PC (Principal Component) which is denoted on Eq. 3 has four elements which are two major factors for application and resource, a set of representative tasks and credit. A *credit* value is decisioned initially as -1 and will be controlled during '*Evaluation step*'(details are in Sect. 4.1.3)

Table 4 presents a set of accumulated profiles for Ex1. Values for the resource factors are determined depending on a monitoring information (Table 3) and each resource element will be converted into a set of scores as described on Table 5. In case of t_{276}(a task profile that id is 276) *cf.* the first row

Table 4 Example profiles of autodock application

id	*App* factors of t_{id}							r	s	vm_{id}	$T_{exec}(id)$
	fa_1	fa_2	fa_3	fa_4	fa_5	fa_6	fa_7				fr_7
276	1	1	300,000	50	150	270,000	50	Sydney	*Done*	*m3.xlarge*	150
277	2	1	300,000	50	150	270,000	50	Oregon	*Failed*	*m1.small*	126
...	...							...			
572	3	1	600,000	50	150	27,000	100	Tokyo	*Done*	*t1.micro*	290
573	4	1	600,000	50	150	27,000	100	Tokyo	*Done*	*m1.large*	232

Table 5 Converted profiles of the above example

id	*App* factors of t_{id}							*Res* factors of t_{id}						s	vm_{id}	$T_{exec}(id)$
	fa_1	fa_2	fa_3	fa_4	fa_5	fa_6	fa_7	fr_1	fr_2	fr_3	fr_4	fr_5	fr_6			fr_7
276	1	1	300,000	50	150	270,000	50	3	0.99	3	3	6	0.98	*Done*	*m3.xlarge*	150
277	2	1	300,000	50	150	270,000	50	−1	0.20	3	5	−1	0.91	*Failed*	*m1.small*	126
...				...				...						...		
572	3	1	600,000	50	150	27,000	100	4	0.96	5	−1	7	0.99	*Done*	*t1.micro*	290
573	4	1	600,000	50	150	27,000	100	4	0.96	5	−1	7	0.99	*Done*	*m1.large*	232

Table 6 Applying PCA to profiles

id	*App* factors of t_{id}							*Res* factors of t_{id}						$T_{exec}(id)$
	fa_1	fa_2	fa_3	fa_4	fa_5	fa_6	fa_7	fr_1	fr_2	fr_3	fr_4	fr_5	fr_6	fr_7
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*

in Table 4, it shows that the task was executed on the site located on *Sydney*. Based on Table 3, it is converted into six values as depicted on (1st row) Table 5.

When the analysis is being performed, the values in profiles are converted to a normalized form by quantile normalization. The final results can be shown on Table 6 and Fig. 2 as scree plot graph. In Fig. 2, X-axis represents each factor (being aligned according to size of *eigenvalue*) and Y-axis is corresponding to *eigenvalue* of each of them.

As a result, we can notice that application and resource factor having the highest eigenvalue among each category are fa_5 and fr_2, respectively (both have rank 1 as depicted in bold underline on Table 6). Accordingly, the result leads to the fact that principal factor for application (fa_{prin}) is fa_5 and resource's one (fr_{prin}) is fr_2. Moreover, eigenvector of a factor (e.g., fa_5) which has the highest eigenvalue among all factors of both categories is used for provisioning. In this example, $EigenVector_5$ is such eigenvector (Eq. 4) and is composed of 13 values (the number of total factors, d) (Table 7).

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

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

Using the vector, we can compute $PCscore$ for all tasks (n tasks) and an example for one task is presented on Eq. 5. After applying Eq. 5 to all tasks (i.e., n times), PCscores are calculated for all the tasks and the results can be plotted as shown on Fig. 3. X-axis of it is score obtained from calculating Eq. 5 using 1st principal factor's eigen vector and

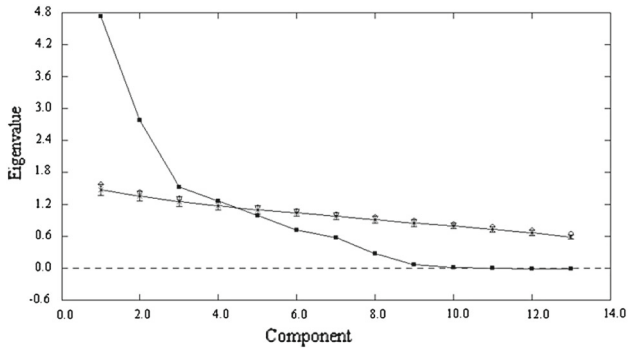


Fig. 2 A scree plot of analysis

Table 7 Notations for PHAP model

-	Property	Description
P_{id}	$[t_{id}, vm_{id}, r, T_{exec}(id), s]$	A profile tuple
t_{id}	$(fa_1, \cdots, fa_n, fr_1, \cdots, fr_m)$	task vector include -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

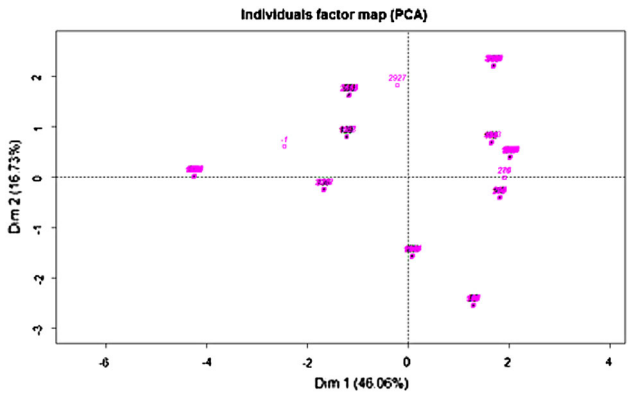


Fig. 3 Plots of PCA scores

Y-axis is that obtained from same equation using 2nd's one. On Fig. 3, it is shown that ten clusters are formed and each of them consists of tasks having similar properties of factors, such as VM instance type. Among the clustered tasks, the one which has the highest score (existing on the right-most part of Fig. 3) is included to $\mathbb{J}_{prin}$ in Eq. 3.

4.1.2 Allocating and provisioning

Using the resulting factors, it offers proper allocation and provisioning. The 'proper' allocation and provisioning is ultimately getting optimal resource (location) and VM which contribute to good performance. To do so, it uses application and resource factor to extract principal factors through relation between the factors and execution time, and gets principal factors for two aspects. And then, for new task submission, it finally refers to one profile having same property of principal application factor as new one's and having high scored resource. Once a job is submitted, it splits into multiple tasks having various arguments. Then system decides which VM is proper and where the VM is deployed. To allocate resource and provision virtual machine, first of all, it checks the existence of PC as shown on Algorithm 1 (line 3).

If there is no PC yet, it carries out **PCA** with a list of the determined factors and recent job traces (line 4). However, if there are no traces yet, we select a cloud site according to Round-Robin order until a system gets enough profiles.

If it has PC through the analysis, **ProfileSelect** function is achieved (line 6) in order to choose appropriate profile to be used for allocation and provisioning.

Algorithm 1 *PHAP* model

1: PC , t_{new}, $PC_{selected} = null$
2: F={*defined factors*}, $\mathbf{t}_{recent}$ is a set of w recent profiles.
3: **if** PC not exists **OR** PC.*credit* < -1 **then**
4: PC =**PCA**(F, $\mathbf{t}_{recent}$);
5: **end if**
6: $P_{selected}$=***ProfileSelect***(PC.fa_{prin}, PC.fr_{prin}, t_{new})
7: $r_{selected} = P_{selected}.r$
8: vm_{new}=$vm(PC.\mathbb{J}_{prin} \cap P_{selected})$
9: // { here, $P_{selected}$= $PC.\mathbb{J}_{prin} \cap P_{selected}$ }
10: **if** *available* vm_{new} on $r_{selected}$ not exists **then**
11: deploy vm_{new} on $r_{selected}$
12: **end if**
13: *Schedule* t_{new}
14: **Evaluate** ($T_{exec}(new)$, $T_{exec}(selected)$);

It requires three arguments: fa_{prin}, fr_{prin}, requested task vector (t_{new}) and the details are described on Algorithm 2. By performing **ProfileSelect** function, it gets one or more profiles ($P_{selected}$) as well as their site information ($r_{selected}$). To decide a type of vm, it uses intersection tasks between representative task set ($\mathbb{J}_{prin}$) of the corresponding PC and the resulted profiles ($P_{selected}$). A vm type for a new task (vm_{new}) is decided by referring the intersection result's one (line 8).

Once a type of vm is decided, it examines the cloud site($r_{selected}$) to check whether a vm being the same type as vm_{new} is available or not (line 10). If it is available, it schedules the task into the vm, otherwise, it deploys vm_{new} to $r_{selected}$ (line 11) and vm will be pulling t_{new} from user's queue after launched. When the task is finished, it is evaluated by calculating difference of execution time between the task (t_{new}) and the referred profile (line 14). By controlling the *credit* according to the above difference, it evaluates the profile to be used by new task. The evaluation step is indicated on Sect. 4.1.3 in detail.

Algorithm 2 ***Profile Select*** Algorithm

Require: $fa_{prin}, fr_{prin}, t_{new}$
1: $\mathbf{P}_{candidate}, \mathbf{P}_{selected} = null$
2: $\mathbf{F} = \{fa_1, \ldots, fa_n, fr_1, \ldots, fr_m\}$where $n + m = d$
$fa_{prin} \in \{fa_1, \ldots, fa_n\}, fr_{prin} \in \{fr_1, \ldots, fr_m\}$
3: $\mathbf{P} \in JobHistory$
4: **repeat**
5: each **P** In *JobHistory*
6: find **P** where P.$t(fa_{prin}) == t_{new}(fa_{prin})$
7: $P_{candidate} = P_{candidate} \cup \mathbf{P}$
8: **until** $\mathbf{P}_{uncheck} == empty$
9: $P_{selected}$=MIN($P_{any}.T_{exec}$)&&MAX($P_{any}.t(fr_{prin})$)
Ensure: $P_{selected}$

In Algorithm 2, it explores principal profile(s) which is used for allocating and provisioning. For every existing profiles, it searches profiles whose principal application factor (fa_{prin}) has the same value with the requested task's one and include the profile in a set of candidate profiles (line 6–7). For Ex1, suppose that the resulting fa_{prin} is $fa_3$and that one of the requested tasks has application arguments t_{new} = [fa_1, ..., fa_7] = [1, 1, 300,000, 50, 150, 270,000, 50]. Then, it searches profile having '300000' of fa_3 among entire traces. After getting candidates, it elects profile(s) which has minimum execution times and has resource value(r) having maximal value of principal resource factor among the candidate (line 9). Suppose that principal resource factor is fr_2(reliability), a location having maximal score of principal resource factor would be $California$ or $Singapore$ according to Table. 3. Finally, it returns the satisfied profile(s) as $P_{selected}$ into line 6 of Algorithm 1.

4.1.3 Evaluating

When a task is completed, a new profile for the task (P_{new}) will be created and it should be put on evaluation step as shown on Fig. 4.

First of all, it checks status of the task. If the status is '*FAILED*', it controls stability of the site decreased and credit of $P_{selected}$ as well. If not, it calculates difference of execution time between the task (P_{new}) and referred one ($P_{selected}$). In case of that the difference is higher than *threshold*, it subtracts 1 to the *credit* of the referred profile. If the gap between two profiles is lower than $Threshold$, it increases *credit* of $P_{selected}$ by 1 and includes the profile of new task into '(Job) Profiling Repository'.

Once *credit* is subtracted, it examines whether the credit value is under -2, or not. If it is, system performs PCA again with the recent traces. If not, it just add new profile on '(Job) Profiling Repository' and finishes the evaluation. In this way, it adapts to the current status of dynamic computing environments and offers reliability on overall processes of this model.

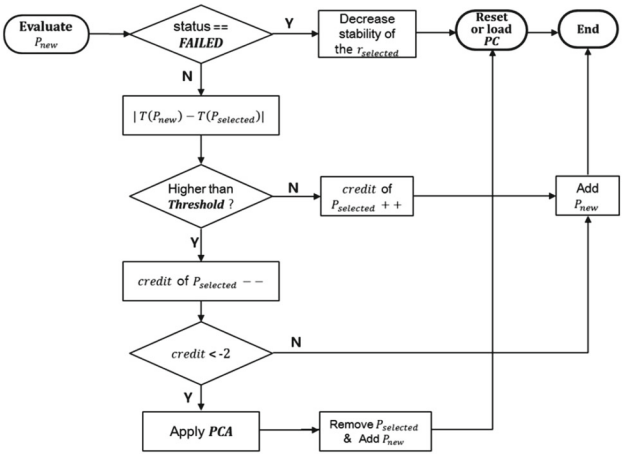


Fig. 4 Adaptive evaluation

5 Experiments

5.1 Experimental setting

We performed this experiment based on measurements over real system which provides management services of HTC jobs and IaaS services, referring to HTCaaS [8]. HTCaaS system offers *Agent-based* job execution service and we had to adjust our scenario in accordance with its *agent-based* concept in order to construct a proper experiment environment. *Agent* in HTCaaS is a pilot-job which pulls tasks from indicated queue and prepares to launch the tasks on submitted site (resource).

For cloud infrastructure, in addition, we adopt Amazon EC2 [10] 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 5 depicts the configured topology.

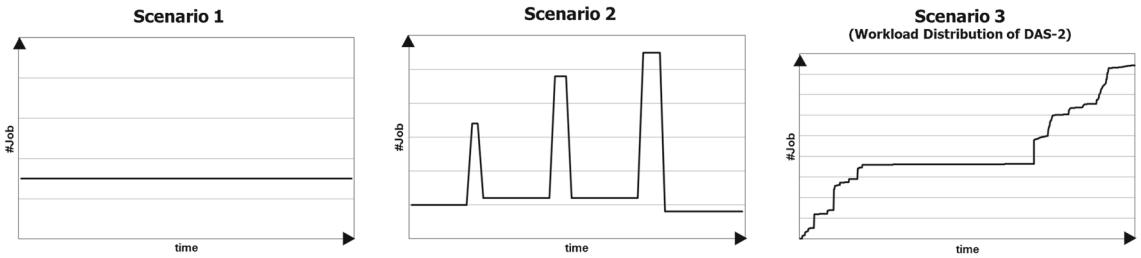
As described on below, we assumed that a service provider is located on Korea (Daejeon) and that each site is connected via WAN. A table under the figure shows distance values and scores for all locations. The distance value refers to the number of hops from submission (Daejeon) and score denotes to performance value as stated before. Here when we apply and implement PCA to HTCaaS, Flanagan Scientific Library [11] is exploited. We run two kinds of real applications which are Autodock and Madgraph5 as previously addressed. In case of Autodock, 10 targets for docking are used in this experiment.

5.2 Scenarios

For performance evaluation, we ran the experiment with several scenarios and compared our model to four models on each scenario. Each scenarios has different and various types of workloads/properties as indicated on the following:

	Virginia	Oregon	California	Ireland	Singapore	Sydney	Tokyo	Sao-paulo
distance	4	5	5	3	2	2	1	4
cvrt_score	1	-1	-1	2	3	3	4	1

Fig. 5 Network topology of Amazon EC2 Site



(a) Scenario 1 **(b)** Scenario 2 **(c)** Scenario3 (DAS-2)

Fig. 6 Arrival patterns for all scenarios

- *Scenario 1* uniformed workload (Fig. 6a)
- *Scenario 2* synthetic workload consisting of uniform and bursty patterns (Fig. 6b)
- *Scenario 3* workload following DAS-2 [12]'s arrival patterns (Fig. 6c)
- *For all scenarios, randomly generated parameter combinations are applied.*

The first scenario is to show an environment with simple patterns of workload without any interferences and the second one is an environment having unexpected and sudden overloads for several times. On the third scenario, DAS-2 [12] workloads which is from GWA (Grid Workload Arhieve) [13] is adopted and its arrival pattern shows increasing tendency throughout the entire time. All patterns of the above scenarios are depicted on Fig. 6a, b, c, respectively. For all of above scenarios, randomly generated application arguments are used for every examples.

We employed Baseline, Random, Round-Robin and Best-Effort methods to compare with our model. Their details are as follows:

- *Baseline* chooses a site in order of distance among available site, but if a task (or *vm*) is failed on a site, it skips the site's turn twice. It deploys *vm* in performance order.
- *Random* selects site and *vm* randomly
- *Round-Robin (R-R)* chooses a site in order of distance and deploy *vm* in order of performance.
- *Best-Effort (B-E)* selects a site having the minimal tasks among entire sites first and deploy a high performance *vm*.

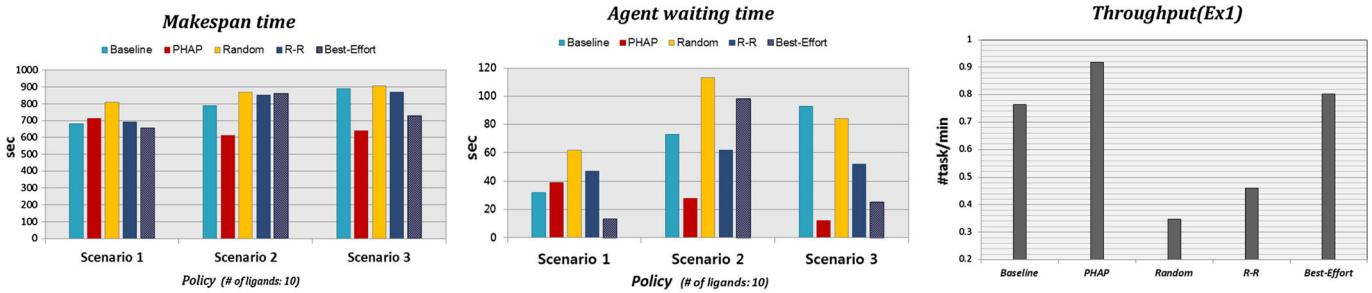
Among these policies, *Baseline* corresponds to a policy which is currently used in HTCaaS for allocating and provisioning *vm*. *Random* is to show the worst case, and *Best-Effort* policy shows the best case in this experiment.

Overall, we ran ten thousand (10^4) number of tasks five times for Ex1 and one million (10^6) tasks five times for Ex2 since many tasks are generated in one job in Madgraph5. Then we've exploited an average of results for performance evaluation. Performance metrics that are used for comparisons, we employ *throughput*, *makespan time*, *waiting time*. In particular, we measured *Agent waiting time* for the *wait time* in accordance with execution paradigm of HTCaaS. It even implies how accurate the decisions are and might include *queuing time*, *preparation time* of agent and *extra-time* (overhead). The queuing time represents the duration of wait from submission to task starting.

A supplementary experiment was performed to explore appropriate size of profile(w) to be used for profiling.

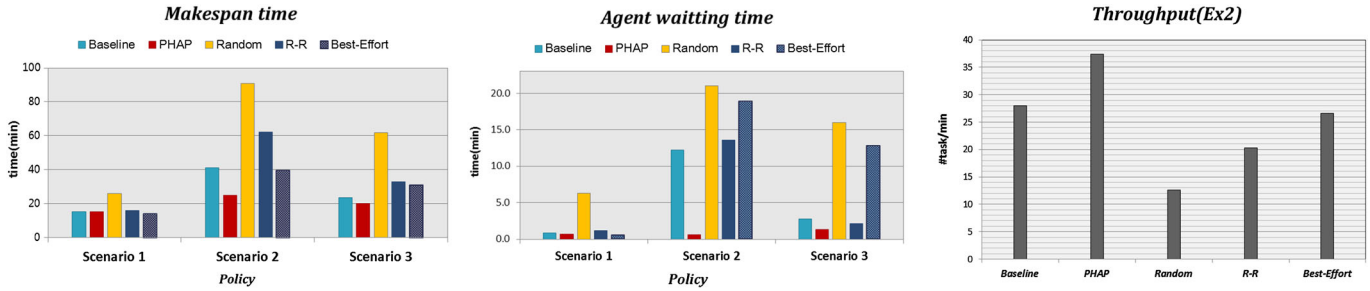
5.3 Results

With respect to the addressed conditions, results of the evaluation are as follows. Figure 7 shows a set of results for Autodock application (Ex1) and Fig. 8 depicts Madgraph5's results (Ex2).



(a) Makespan **(b)** Average wait time **(c)** Throughput

Fig. 7 Experimental results(Ex 1)



(a) Makespan **(b)** Average wait time **(c)** Throughput

Fig. 8 Experimental results(Ex 2)

In Example 1 with *Scenario 1* having no burst workloads, 'B-E' has the best performance of makespan time (approximately 640 s on average) and 'Random' has the worst performance resulting in approximately 810 s on average. Meanwhile, other three policies including our model have led to around 700 s. In Example 2 where the number of tasks per job has range from 500 to 1500, with the 1st scenario, B-E resulting in the best performance took approximately 14 min and Random took nearly 26 min on average. Other three policies (Baseline, PHAP, R-R) have brought on results ranging from 15 to 16 min.

In *Scenario 2* which has several burst loads, results of Example 1 show that overall increments of makespan time have occurred on almost policies except 'PHAP' in which a mean makespan time is shorter than other policies with a maximum of 260 s. It seems to attribute relatively short waiting time to the above result as depicted on Fig. 7b. Example 2's results show much more increments comparing to Example 1. For Example 2, the average number of tasks per one job is higher than those for Example 1, and it appears that the obtained result was driven by the effect of the amplification of overload (the worst is Random which had resulted in nearly 3 times more of scenario 1's). According to an analysis of logs for these experiments, the increments were mainly caused by problem of heavy concentration on a specific site (i.e., high density on a specific site). Results of waiting time for other four policies(Baseline, Random, R-R, B-E) also show prolonged durations which are caused by accumulated waiting requests on specific sites (Fig. 8b).

On the *3rd scenario* which has gradually increasing pattern of workload, Example 1's results turn out increments of the time than the second scenario on almost policies except two policies, PHAP and B-E. In case of B-E, according to analysis of logs, it is shown that it had an optimized performance on the first half of the entire period and had delays on the latter part caused by accumulated loads.

In Example 2 on the third scenario, overall durations on all policies tend to decrease more than the second scenario's unlike Example 1 and PHAP model led to the shortest makespan time among them. Similarly, the result presents that there have been no delays (Fig. 8a) nor re-submissions on PHAP model in contrast with other policies (Baseline, Random, R-R, B-E). Accordingly, '*Agent waiting time*' has resulted in similar patterns with makespan time where B-E led to minimum waiting time and Random has the longest waiting time (Figs. 7b, 8b). Furthermore, it is able to reduce waiting time relatively as well as overhead, as shown on Fig. 8b. When it targets the higher number of tasks like Example 2, it could reduce the waiting time since it has been utilizing more traces and contribute to more clear results. 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 as described on Figs. 7c and 8c. It turned out that it has more throughput than B-E(which is best case) of 0.12 in Example 1 and of 11 in Example 2.

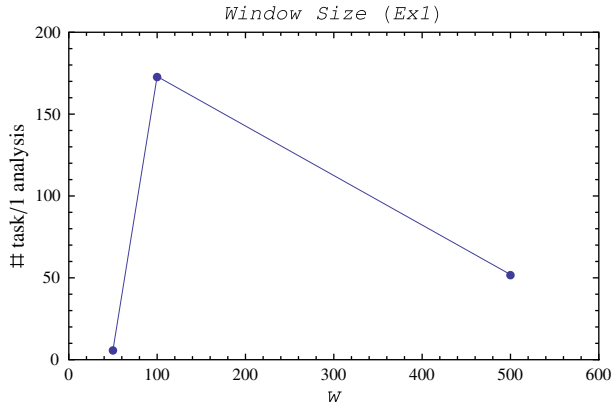


Fig. 9 Optimal window size for autodock(Ex1)

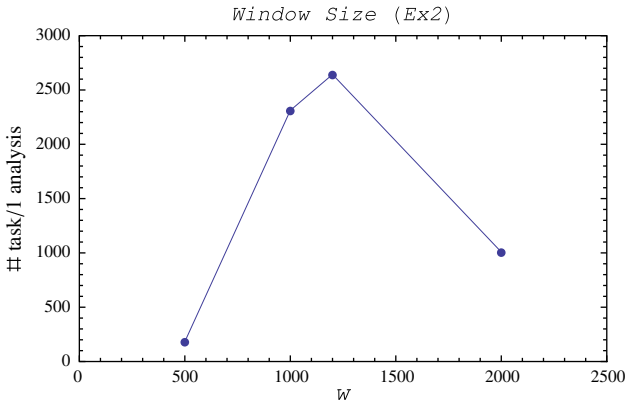


Fig. 10 Optimal window size for madgraph5(Ex2)

In conclusion, we demonstrated that our proposed model is highly efficient on varying patterns of workloads as well as computing models in which numerous tasks are generated.

Additionally 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 results are depicted on Figs. 9 and 10 and show that appropriate profile size to be used in analysis is approximately 100 in Example 1 and 1200 in Example 2 where the system affords to cover maximum number of tasks in an analysis. By making use of the measured size, it can result in reliable analysis for this model.

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 analyse the profiles of jobs, we applied PCA technique as a statistical method to elect the effective factors. They are employed for selecting cloud site and deciding proper virtual machine type.

The performance evaluation is achieved on HTCaaS system which is an agent-based multi-level scheduling system and its results shows that our model can improve overall performance of high throughput computing applications compared with other policies such as Random, Round-Robin and Best-Effort. The results turned out that our model is capable of improving overall throughput by reducing makespan time, and that it is more beneficial on an environment which has varying pattern of workloads and deals with plenty of tasks in general.

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. In addition, we will improve this model to reduce overall cost for public cloud by adding a cost factor.

Acknowledgments S.Y Kim thanks S.-h. Nam for useful comments and supports. This research was supported in part by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT and Future Planning (NRF-2013R1A1A3007866)

References

1. Wang, L., Zhan, J., Shi, W.: In cloud, can scientific communities benefit from the economies of scale? TPDS **99**, 1 (2011)
2. Wang, X.Y., et al.: Appliance-based autonomic provisioning framework for virtualized outsourcing data center. In: Proceedings of the Fourth International Conference on Autonomic Computing, p. 29 (2007).
3. Li, H., Groep, D., Wolters, L.: Efficient response time predictions by exploiting application and resource state similarities, In Proceedings of the 6th IEEE/ACM International Workshop on Grid Computing. IEEE Computer Society, pp. 234–241 (2005).
4. Urgaonkar, B., Shenoy, P,. and Roscoe, T.: Resource overbooking and application profiling in a shared Internet hosting platform. ACM Trans. Internet Technol. 9, 1, Article 1 (February 2009), pp. 45. 2009.
5. Raicu, I., Foster, I.T., and Yong Z.: Many-task computing for grids and supercomputers", MTAGS 2008. In: Workshop on Many-Task Computing on Grids and Supercomputers, pp. 1–11 (2008).
6. Morris, G.M., Goodsell, D.S., Halliday, R.S., Huey, R., Hart, W.E., Belew, R.K., Olson, A.J.: Automated docking using a lamarckian genetic algorithm and and empirical binding free energy function. J. Comput. Chem. **19**, 1639–1662 (1998)
7. Alwall, J., Herquet, M., Maltoni, F., Mattelaer, O., Stelzer, T.: MadGraph 5: going beyond. J. High Energy Phys. **6**, 1–40 (2011)
8. Rho, S., Kim, S., Kim, S., Kim, S., Kim, J.-S., and Hwang, S.: 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 (SC'12), November (2012).
9. Jolliffe, I.T.: Principal Component Analysis (PCA), Springer Series in Statistics., 2nd edn. Springer-Verlag, New York (2002)
10. Amazon EC2 (Elastic Compute Cloud), http://aws.amazon.com/ec2. Accessed 12 April 2014
11. Flanagan Scientific Library, http://www.ee.ucl.ac.uk/~mflanaga/java/. Accessed 12 April 2014
12. DAS2-Grid, http://cs.vu.nl/das2. Accessed 12 April 2014
13. Grid Workload Archive (GWA), http://gwa.ewi.tudelft.nl/. Accessed 12 April 2014

Seoyoung Kim is a researcher in National Institute of Supercomputing and Networking (NISN) at KISTI (Korea institute of Science and Technology Information). She received her Bachelors and Masters degree from Sookmyung Women's University in 2010 and 2012, respectively. She has worked in NISN, KISTI from 2012. Her research interests focus on meta-scheduling in distributed computing systems, particularly (on) High-Throughput Computing, Many-Task Computing.

Jik-Soo Kim received a Ph.D. in Computer Science from University of Maryland at College Park in 2009. He is currently a Senior Research Scientist in the National Institute of Supercomputing and Networking at KISTI (Korea Institute of Science and Technology Information). Dr Kim's primary interests are in the design and analysis of distributed computing infrastructures to support High-Throughput Computing, Many-Task Computing and Cloud Computing.

Soonwook Hwang is a team leader in Korea Institute of Science and Technology Information (KISTI), leading the research and development of value-added technologies and services mainly based on grid and cloud technologies for the realization of e-Science paradigm in many areas ranging from high energy physics to medical physics and life science. He has been responsible for the development of the AMGA metadata system and its production deployment for the Belle II experiment at KEK in Japan. He is also leading the development of a large-scale HTC Problem Solving Environment called HTCaaS by fully exploiting the national distributed supercomputing infrastucture called PLSI in Korea. Dr Hwang received his

BS and MS degree from Seoul National University in Korea and got his PhD in Computer Science from the University of Southern California.

Yoonhee Kim is a professor in the Department of Computer Science, Sookmyung Women's University. She received her Bachelors degree from Sookmyung Women's University in 1991, her Masters 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.