Cloud Provisioning for Workflow Application with Deadline using Discrete PSO

Nuttapong Netjinda[1],
Tiranee Achalakul[2], and **Booncharoen Sirinaovakul**[3], Non-members

ABSTRACT

The need of cloud consumers to optimize all options offered by cloud provider has been rapidly arisen during the recent years. The consideration involves the appropriate number of VMs must be purchased along with the allocation of supporting resources. Moreover, commercial clouds may have many different purchasing options. Finding optimal provisioning solutions is thus an NP-hard problem. Currently, there are many research works discussing the cloud provisioning cost optimization. However, most of the works mainly concerned with task scheduling. In this paper, we proposed a new framework where number of purchased instance, instance type, purchasing options, and task scheduling are considered within an optimization process. The Particle Swarm Optimization (PSO) technique is used to find the optimal solution. The initial results show a promising performance in both the perspectives of the total cost and fitness convergence. The designed system provides the solutions of purchasing options with optimum budget for any specified workflow-based application based on the required performance.

Keywords: Cloud Computing, Cloud Provisioning, Cost Optimization, Workflow Application, Deadline Constraint, Particle Swarm Optimization

1. INTRODUCTION

Cloud computing service has gained popularity in various application domains during the recent years. Cloud computing refers to the computing resources delivered as services over the Internet. The hardware and systems software in the datacenters provided those services is called Cloud. Cloud infrastructure can serve many purposes [5]. Firstly, it can be used to serve internet-based services. Instead of owning and maintaining web servers or database servers, a company can purchase virtual servers on the cloud. Secondly, cloud can be used as a High Performance Computing (HPC) platform. Cloud consumers can purchase virtual machines that yield the required performance to execute the large scale scientific workflow application.

The price of cloud computing service is scalable and adjustable according the service options. To get the most advantage form cloud computing service; there are many factors that the cloud customer has to consider. These factors are number of purchased instances, instance type, purchasing options, and task scheduling. Cloud customer must balance all factors with the required performance to execute the workflow application.

Usually, Cloud providers offer 2 options in purchasing cloud computing instances, on-demand instance and reserved instance. The usage cost for the on-demand instance is charged based on a per-hour basis while the usage cost for reserved instance is charged in advance for a certain amount of service time beside the per-hour cost. However, the per-hour cost of reserved instance is cheaper than the per-hour cost of on-demand instance. When comparing to the on-demand plan, the best advance reservation plan is difficult to achieve due to the uncertainty of cloud consumer's demand in the future and providers' resource prices. Decision on whether to purchase cloud instance as reserved or on-demand instance is another importance policy. A good decision leads to reduction on total cost. In addition, cloud providers often offer many type of instances with different computational power. Consumer can then select the instance types that match their requirement.

There are 2 types of problem for finding the optimal solution of cloud provisioning cost, scheduling and cost modelling.

To find the optimal cost of cloud computing, Chaisiri et. al. [7] proposed an optimal cloud resource provisioning (OCRP) algorithm. The stochastic programming model is formulated to find the total resource provisioning cost. Byun et al. [6] proposed a partition time balance schedule (PBTS) to optimize the cost of executing workflow application on a cloud platform. Their research focuses on scheduling tasks to minimize a number of cloud instances in each time slot.

The cost-optimal of application with many tasks is a complex problem especially for the application with dependency relationship. For cloud provisioning problem, Chen and Wang [8] proposed PSO for task

Manuscript received on July 20, 2012 ; revised on November 16, 2012.

[1,2] The authors are with Department of Computer Engineering, King Mongkut's University of Technology Thonburi, Bangkok, Thailand, E-mail: nuttapong.net@gmail.com, tiranee@cpe.kmutt.ac.th and boon@kmutt.ac.th

scheduling in heterogeneous grid. Their objective is to minimize application completion time. Many PSO based algorithms have been proposed to minimize cost of executing workflow application on cloud computing [14, 16]. All the above approaches are focused on scheduling tasks in workflow application to be executed on cloud instance. In their works, the number and computation power of cloud instance are assumed to be static.

In order to take the instance, purchasing options and task scheduling all together into account, the problem becomes NP-hard. In the real world, prior to the execution a large scale scientific application project, the number of instance to be purchased has to be decided on. Also, in each purchased instance, decision on instance type has to be made to serve the required computational power. The decision on whether to purchase cloud instance as reserved or on-demand instance is another importance policy. A good decision leads to reduction on total cost while the purchased cloud instances can serve the required performance. To address these problems, we propose cost optimization in cloud provisioning using PSO.

This research focuses on the optimization of cloud instances purchasing for large scale scientific workflow executions. Types and numbers of instances to be purchased are considered in our framework along with the execution dateline. We extended the experiments from our previous work [13]. We have included the experiments on finding the convergence speed and more details on the effect of deadline. The results of our work can help the cloud consumer to decide whether to purchase on-demand or reserved instances. A good decision can also lead to cost saving, while maintain the required performance.

In optimization, heuristic algorithms were designed to deal with specific problems. If the scope of problem is changed, the algorithm has to be redesigned. Furthermore, heuristic algorithm may not produce the optimal solution. Metaheuristic algorithms were invented to deal with generic problem [11]. This kind of algorithm provides both exploration and exploitation in solution searching. With the well balance in exploration and exploitation, metaheuristic algorithm yields the excellent results. The examples of popular metaheuristic algorithms nowadays are Artificial Ant Colony, Particle Swarm Optimization, and Artificial Bee Colony algorithms. Banharnsakun, Sirinaovakul and Achalakul [2] proposed an algorithm to solve job shop scheduling problem by using Best-so-far ABC [3]. The purpose of their research is to find the best scheduling solution in the production system.

Among metaheuristic algorithms, there are both algorithms for combinatorial problems and for numerical problems. For example, Artificial Ant Colony algorithm was designed to deal with combinatorial problems and Particle Swarm Optimization algorithm was designed to deal with numerical problems. Particle Swarm Optimization provides high performance in optimization. Although it was designed to deal with numerical problems at first, it can be modified to deal with combinatorial problems and yields the better performance than Artificial Ant Colony [9].

The research contributes to cloud computing provisioning cost optimization system which is considered to be combinatorial problem. The system uses Particle Swarm Optimization to minimize the total cost of using cloud. The framework takes workflow detail, task dependency and task execution time, as inputs. Appropriate instance number, instance type, purchasing type, and task schedule which produce minimum cost, is then achieved as a result.

The rest of this paper is organized as follows. Section 2 describes the cost model. Section 3 describes original Particle Swarm Optimization. Section 4 describes the decoding scheme to be applied to Particle Swarm Optimization in our problem context. Section 5 describes our experiments. Section 6 shows results from our experiments. Finally, Section 7 concludes our work.

2. COST MODEL

In order to find the optimal solution, cost model is required to be an objective function. It is used for calculating the cost of purchasing cloud provisioning cost. In this research, the cost model consists of 2 major parts, on-demand and reserved cost. The model is designed whether the instances are purchased with on-demand pricing and/or reserved pricing. The detail of cost model is shown in equation (1).

$$Cost = \sum_{m=1}^{M} c_d^m t^m x_d^m + \sum_{m=1}^{M} (f_r^m + c_r^m t^m) x_r^m \quad (1)$$

From equation (1), x_d^m and x_r^m are the indicator variables that indicate the instance m was purchased as on-demand (x_d^m) or reserved instance (x_r^m). The variable c_d^m and c_r^m are the per-hour cost of on-demand instance and reserved instance respectively. If the consumer purchases only reserved instance then the value of x_d^m and c_d^m are zero. On the other hand, if the consumer purchases only on-demand instance then the value of x_r^m and c_r^m are zero. However, the instances can be purchased as on-demand, reserved or combination of on-demand and reserved. The variable f_r^m is the cost that cloud consumer has to pay the provider in advance for reserved instance. Lastly, the variable t^m is the time required for each machine to run the tasks.

The decision of all alternatives in equation (1) must be satisfied the constraint in equation (2). That is time span must be shorter or equal to the deadline. In other words, the executing time of the workflow must be finished on time or before the deadline. This time span is the time difference between the start and finish of a sequence of tasks.

$$\textit{time span} \leq \textit{Deadline} \quad (2)$$

To find the optimal solution, the cost function must guide the search process in the problem space to find the optimum solution. Therefore, the cost of the alternative solution that cannot meet the deadline constraint in equation (2) must pay the penalty. In the case that deadline constraint is satisfied, we use the cost from equation (1) as the fitness value. Otherwise, we apply the penalty cost to equation (1). Therefore, the fitness value for the alternative solution that the execution time cannot meet the constraint is changed to equation (3) as shown below.

$$\textit{Fitness} = \textit{Cost} \times 2 \times \frac{\textit{Time Span}}{\textit{Deadline}} \quad (3)$$

The variable *Cost* in equation (3) is the cost which is calculated from equation (1). The *Time Span* is the time required for executing the workflow.

3. PARTICLE SWARM OPTIMIZATION

To provide the good balance between exploration and exploitation in solution searching, we use Particle Swarm Optimization as the base algorithm in our work. Particle Swarm Optimization (PSO) [12, 15] is a variation of swarm intelligence. PSO search for the solution which is multi-dimensions real value by simulating the particle moving around the solution space. The position of particle in solution space represents a solution for the problem. Each particle has its own velocity to update its position. Each particle remembers the personal best which is the best solution that particle ever found. Among the personal best from all particles, the best one is remembered as the global best. In each iteration, the particles' velocity is updated using equation (4).

$$v_k^{t+1} = wv_k^t + c_1r_1(p_k^t - x_k^t) + c_2r_2(p_g^t - x_k^t) \quad (4)$$

Where w, c_1, and c_2 are the constant factors, r_1 and r_2 are the random real number between 0-1. For particle k after t iterations, v_k^t is its current velocity. x_k^t is its current position. p_k^t is its personal best. p_g^t is the global best after t iterations. After updating velocity, the particles' position is updated using equation (5).

$$x_k^{t+1} = x_k^t + v_k^{t+1} \quad (5)$$

The particles' velocity and position updating iteration continues until the termination criterion is met. The global best from the last iteration is used as the solution.

PSO can be directly dealt with problem which solution can be represented as real value. For the problem with discrete solution, PSO has to be modified. One method is changing the particle into the vector of discrete value instead of real value [10]. This method changes the definition of operator in equation (4) and (5) to be able to handle integer vector. The other method is decoding from real value into integer [8] which we used in this paper.

4. PARTICLE DECODING

PSO algorithm is originally designed for solving the numerical problem. However, the solution of cloud provisioning cost optimization problem is discrete alternatives which are represented with integer. Therefore, real values in PSO's particle have to be decoded into integers to represent a solution. The decoding scheme used in this paper is adapted from [8] which the real values in PSO's particle are rounded up to be string of integers. Our particle string is designed to represent the decision variable with real values. After the PSO optimal value searching process, these values are rounded up to be integers for further decision. The representation decoding consists of two processes. First, PSO's particle is a set of string of real value which represents the vector of decision. This string of real value is divided into sections. Each section is used to represent decisions for each part of the problem. Each section contains many dimensions. Each dimension has many alternatives. Second, the real value in each dimension is decoded into integer by multiplying with the number of alternatives and then round up the multiplication result to be integer. For example, a dimension of string variable represents selection of two alternatives; these two alternatives are represented as integer 1 and 2. If the real value of the string variable is 0.3, multiplying with 2 yields 0.6. We round up 0.6 and get 1 which means that we select the first alternative. If the real value of the string variable is 0.7, multiplying with 2 yields 1.4. We round up 1.4 and get 2. Therefore, we select the second alternative. Using this decoding scheme, we can use PSO without modification on particle updating process.

To represent the solution, PSO's particle which is the decision vector (or string of variables) is divided into 5 sections. The first section represents the total number of instances to be purchased. The second section represents the selected type of each instance. The third section represents the purchasing type of each instance. The fourth section represents the task assignment for scheduling. The fifth section represents the task priority for scheduling. In our problem, we design the real value in all dimensions to range from 0-1. These values are used to indicate the alternative that will be selected in the dimension. The example of PSO's particle is shown in Fig. 1.

The structure of solution string is designed as follow. The first variable represents number of purchased instances. The number of the variables of instance type and purchasing type is equal to the max number of purchased instances. The max number of purchased instances has to be specified in advance. If

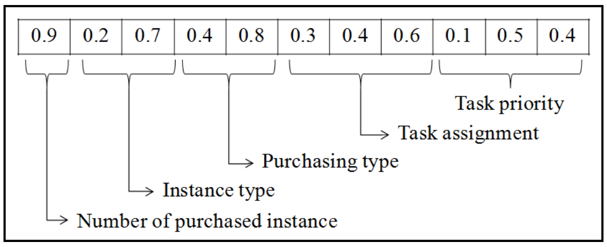

Fig.1: *Example of PSO Particle.*

the number of purchased instances is less than max number, the variables beyond that number are ignored. The number of variables for task assignment and task priority is equal to the number of tasks that we want to execute in the workflow. Suppose that there are m max purchased instances and x tasks. Therefore, the total number of variables of the solution string is:

1+m+m+x+x = 1+2m+2x

If m=2 and x=3 then the total number of variables are:

1+2*2+2*3 = 11

To illustrate our decoding scheme for PSO's particle in Fig. 1, we consider the problem with the following parameters. Suppose that consumer sets max number of purchased instance to 2. There are 3 alternatives for instance type; small, large and extra-large instance and 2 alternatives for purchasing type; reserved and on-demand. There are 3 tasks in the workflow. Then, pseudo-code for decoding scheme can be written as shown in Fig. 2.

```
MAX_INSTANCE = 2, TYPE_NUM = 3, PURCHASE_NUM = 2,
TASK_NUM = 3
instnum = ceil(x[0]*MAX_INSTANCE)
instype = [ceil(x[i]*TYPE_NUM) for i = 1
           to MAX_INSTANCE]
purtype = [ceil(x[i]*PURCHASE_NUM)
           for i = MAX_INSTANCE+1 to 2*MAX_INSTANCE]
assign = [ceil(x[i]*instnum) for i = 2*MAX_INSTANCE+1
          to 2*MAX_INSTANCE+TASK_NUM]
priority = [x[i] for i = 2*MAX_INSTANCE+TASK_NUM+1
            to 2*MAX_INSTANCE+2*TASK_NUM]
```

Fig.2: *Pseudo-code for Particle Decoding.*

The first section, total number of instances, consists of one dimension. This dimension represents the number of instances to be purchased which is limited to max number of purchased instances. Supposed that the first dimension has 2 alternatives, therefore the value in the first dimension, 0.9, is multiplied by 2 (MAX_INSTANCE) which yields 1.8 and then rounded up to 2. This represents 2 instances.

The second, instance type, has the number of dimensions equal to max number of purchased instance (MAX_INSTANCE). Each dimension in these sections represents instance type of the corresponding instance. There are 3 alternatives for each dimension. Therefore, the real values in instance type section are multiplied by 3 (TYPE_NUM) which yield 0.6 and 2.1 and then rounded up to 1 and 3 respectively. These represent that the first instance is small instance and the second instance is extra-large instance. If the number of instances to be purchased is less than max number of purchased instances, the latter dimensions will be ignored.

The third sections, purchasing type, also has the number of dimensions equal to max number of purchased instance (MAX_INSTANCE). Each dimension in these sections represents purchasing type of the corresponding instance. Since there are 2 alternatives for each dimension in this section, the real values are multiplied by 2 (PURCHASE_NUM) which yield 0.8 and 1.6 and then rounded up to 1 and 2 respectively. This represents the purchasing type of the first and the second instance which are reserved and on-demand respectively. Like section 2, if the number of instances to be purchased is less than max number of purchased instances, the latter dimensions will be ignored.

The fourth section, task assignment, has the number of dimensions equal to the number of tasks in workflow (TASK_NUM). Each dimension in this section represents which instance that task is assigned to. From this example, the real values in this section are multiplied by 2 alternatives (instnum) which is the number of purchased instances we get from first section. Multiplication yields 0.6, 0.8, and 1.2 and then rounded up to 1, 1, and 2 respectively. This represents that the first and the second tasks are assigned to the first instance while the third task is assigned to the second instance.

The last section, task priority, has the number of dimensions equal to the number of tasks in workflow (TASK_NUM). The priorities in this section are used to define the order of execution of the tasks which were assigned to the same instance. The task with highest priority will be executed first if it has no preceding task or all preceding tasks were done. Otherwise, it has to wait until all the preceding tasks are finished. In this example, the first and the second tasks are assigned to the same instance. We give the higher priority to the higher value, so, the second task will be executed before the first task if this does not violate their dependency.

Using this decoding scheme, we can represent selections of number of instances to be purchased, instance type, purchasing type, and task assignment while PSO's particle is still consisted of real values. Another benefit is that the number of alternatives for task assignment can be changed as the number of purchased instances has changed.

To conclude the process of our system, pseudo-code of our PSO is shown in Fig. 3. The process starts with generating particle population. Each particle position is initialized by creating a vector of length 2*MAX_INSTANCE+2*TASK_NUM+1. Each dimension in the particle position is randomly assigned the real value between 0 and 1. Velocity of each particle is initialized by random. The velocity vector has the same length as the particle position. We limit the range of these real values to -0.2 and 0.2. The particle position is then decoded into integer as explained above. We used these integers to represent number of instance, instance type, purchasing type, task assignment. We calculate execution time on each purchased instance and total time span from instance type, task assignment and task priority. The number of purchased instances, instance type, purchasing type, and execution time are used by equation (1) to calculate the total cost. If total time span does not exceed the given deadline, we take this total cost as particle fitness. Otherwise, we calculate particle fitness from equation (3). All particles take their initial position as their personal best. The particle with the lowest fitness is used as global best. All particle velocities and positions are updated by equation (4) and (5) respectively. Their positions are decoded and used for calculating their new fitness. If the new fitness is lower than its personal best fitness, the new position is used as new personal best. After all particle velocities and positions are updated, the personal best with the lowest fitness is used as new global best. Particle updating iteration continues until it reaches the max number of iteration (MAX_ITERATION). The global best from the last iteration is then used as the solution.

5. EXPERIMENT

The system is tested on various kinds of workflow structure which provided by Bharati et al. [4]. These workflows are Cybershake, Inspiral, Montage, and Sipht workflow. For each run, there are 100 tasks in each workflow in our experiment. The experiments are tested on the combination of 2 deadlines and 2 numbers of executions per year. We limited the number of purchased cloud instances to 10 instances. The number of PSO's particles on each workflow is assigned to 20 particles and run for 20 times. The means of the total cost are then calculated. We take pricing model as provided on Amazon EC2 [1] for the cost of instance types. Only the standard instance type is used in our experiments. Standard instance type provides 3 alternatives of instance types. They are small instance, large instance, and extra-large instance. The costs of these instance types are shown in Table 1.

Table 1: *Cloud Computing Instance Costs.*

Instance type	Reserved		On-demand
	Per-term ($)	Per-hour ($)	Per-hour($)
Small	97.5	0.07	0.12
Large	390	0.28	0.48
Extra-large	780	0.56	0.96

For task execution time on each type of instance, we estimate by using time factor as shown in Table 2. These factors are set to estimate execution time instead of using real execution time. We assume that extra-large instance which is the most powerful instance in our experiments can finish each task within the time provided in workflow. Therefore, we set the factor for this type to 1. For large instance and small instance which are less powerful than extra-large instance, we assume that they can finish each task within the time as 30% and 60% higher than extra-large instance. Therefore, we set the factor for this type to 1.3 and 1.6 respectively. These factors will be multiplied to the execution times from the input workflow to estimate execution time of the respective instance type. These estimated execution times are used to find the time span of the work flow.

Table 2: *Cloud Computing Instance Costs.*

Instance	Time factor
Small	1.6
large	1.3
Extra large	1

For simplification, we did our experiments on the following assumptions. Tasks in the workflow do not need any overhead time to be executed. The per-hour cost is calculated from the summation of execution time of that instance over a year. We assume that there is no data transfer among tasks in workflow. We also ignore the cost from transferring data in and out of the cloud since this cost does not depend on the instance type or purchasing type.

Since our work deals with the problem with a specific scope, there is no competitive work which covers exactly the same problem scope. The problem scope in our work includes instance type selection, purchasing type selection, and task scheduling. For evaluation, we benchmark our results against the lower bound values which is described in [6] as well as our method for estimating the lower bound values and the method described in [6] is also similar. Lower bound values are calculated from equation (1) with some relaxation on parameters. We ignore task dependencies within workflows. We select only one type of instance for each run time. The total execution time for that type of instance is then divided by deadline to find the number of instances. We calculate the total cost and take the lowest one from the following purchasing case; all instances are on-demand, all instances are reserved, and all but last instance are reserved and last instance which takes the least execution time is on-demand. Finally, we compared total cost from every instance type and select the lowest cost to be lower bound value. This calculation yields the lower bound value because it minimizes the idle time of purchased instance by ignoring task dependency. Thus, it yields the minimum number of instances to finish

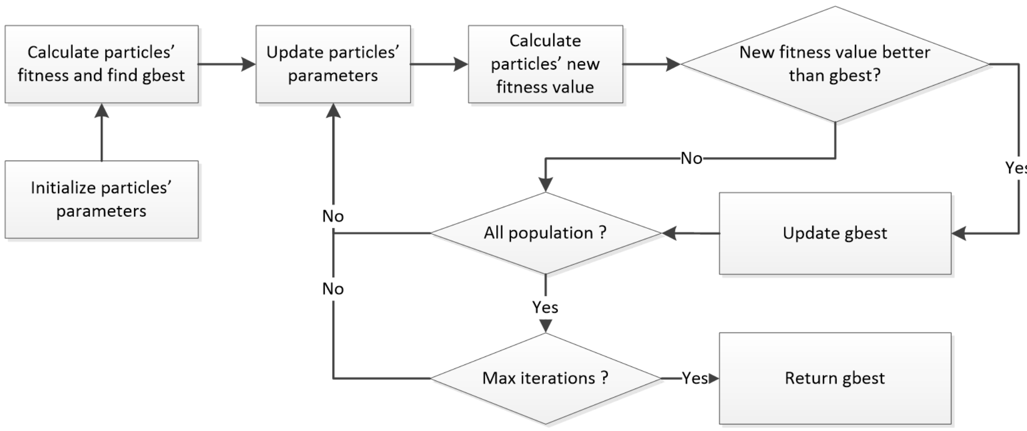
Calculate particles' fitness and find gbest
Update particles' parameters
Calculate particles' new fitness value
New fitness value better than gbest?
Initialize particles' parameters
No
Yes
No
All population ?
Update gbest
No
Yes
Max iterations ?
Yes
Return gbest

Fig.3: *PSO for Cloud Provisioning.*

the given workflow within the given deadline. We take this lower bound value as the ideal total optimum cost.

6. RESULTS

Table 3 shows the results of the experiment as described above. As we will see from the table, the total costs get lower as we increase deadline. Relaxation on deadline for scheduling gives more flexibility in scheduling which leads to the lower cost. As we double the number of execution times per year, the total costs proportion are less than double for every workflow. This reflects that the algorithm yields different alternatives in purchasing when usage behaviour has changed. When we compare total cost to lower bound value, we found that the results from CyberShake and Montage experiments are close to the respective lower bound value. The percent differences between total costs and lower bounds are lower than 10%. For Inspiral and Sipht experiments, the differences between total costs and lower bounds are higher than 10% since their workflow dependency are more complex. These differences are caused by the complexity of the workflow structure. The tasks in Cybershake and Montage can be distributed more than those in Inspiral and Sipht. This makes Cybershake and Montage more flexible in scheduling than Inspiral and Sipht. When we increase deadline of Inspiral and Sipht, the percent differences are reduced because we give more flexibility to the scheduling.

We did more experiments on Sipht workflow by varying deadline. The results in Fig. 4 show the effect of increasing deadline in more detail. When the deadline is low, the difference between total cost and lower bound value is high. When we increase the deadline, the difference becomes lower. With deadline at 5.5 hours, the percentage of difference is almost zero. However, the deadline beyond 5.5 hours might yields the higher percentage. This due to increasing deadline also makes the lower bound value become lower. Decreasing of deadline might not been consistence to the optimal solution. The lower bound value might get away from the optimal solution after this point.

To illustrate performance of PSO on cloud computing provisioning cost optimization, we plot the graphs of means of global best's fitness value versus iteration. We have plotted the graph of the first case of each experiment.

Fig. 5-Fig. 8 show the convergence of fitness value in each experiment. Fitness values decrease rapidly within first 50 iterations. Fitness values decrease slowly or stable after 100th iteration.

For Cybershake (Fig. 5) and Montage (Fig. 7) experiments, the fitness values will be converged at 100th and 50th iteration respectively. This is consistent with the results in Table 3 that total costs are very close to the respective lower bound values. The solutions found are optimal or close to optimal solution and cannot be improved further. For Inspiral (Fig. 6) and Sipht (Fig. 8) experiments, fitness values are still getting lower after 200th iteration. Thus, the gaps between total costs and respective lower bound values can be reduced further if we let our system work for more iteration. This might be the effect from the structure of Inspiral and Sipht workflow. The task assignment of these 2 workflows is harder to be optimized than of Cybershake and Montage workflow. This makes PSO converge slower in Inspiral and Sipht experiment.

7. CONCLUSIONS

This work can be useful to cloud consumers in purchasing cloud computing service for scientific workflow execution. Since the cloud provider offer many purchasing options to consumers, cloud consumers have to decide the number of instances, type of each instance, and purchasing type of each instance. The solutions from our work make cloud consumers able to decide those options at minimum cost. The solution

Table 3: *Cloud Computing Instance Costs.*

Workflow	Execution per year (times)	Deadline (hrs.)	Total cost ($)	Lower bound ($)	%
CyberShake	4,000	0.6	683.38	641.03	6.61
		0.8	633.10	595.18	6.37
		1	591.86	586.53	1.43
	8,000	0.6	1184.33	1087.05	8.95
		0.8	1086.36	995.36	9.14
		1	1002.56	995.36	0.72
Inspiral	1,000	2.5	1266.74	1038.78	21.94
		3	1088.16	963.78	12.91
		3.5	1060.90	946.58	12.08
	2,000	2.5	2169.17	1698.16	27.74
		3	2018.19	1635.06	23.43
		3.5	1782.34	1600.66	11.35
Montage	6,000	0.2	352.03	345.39	1.92
		0.3	348.94	345.39	1.03
		0.4	345.39	322.89	6.97
	12,000	0.2	685.02	645.78	6.08
		0.3	599.95	597.95	0.33
		0.4	590.85	548.28	7.76
Sipht	1,000	2.5	970.34	844.42	14.91
		4	804.70	735.70	9.38
		5	760.53	735.70	3.38
	2,000	2.5	1689.81	1396.34	21.02
		4	1358.96	1276.41	6.47
		5	1286.29	1276.41	0.77

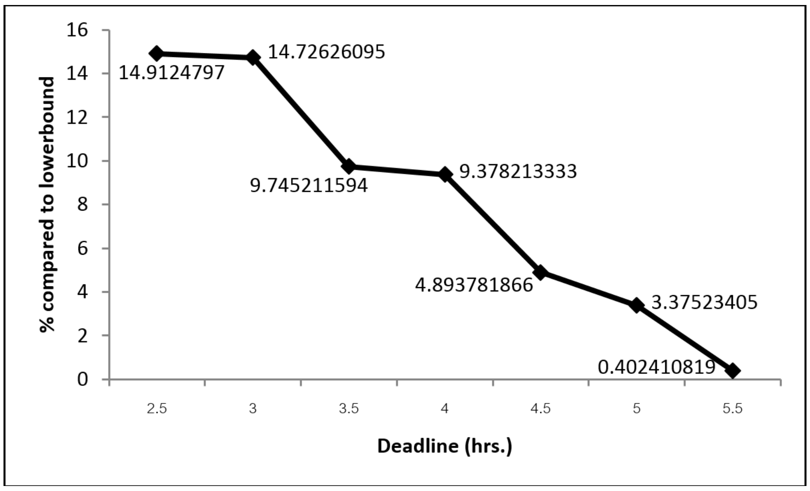

Fig.4: *Percentage of Total Cost Compared to Lowerbound value.*

also provides task scheduling to execute the given scientific workflow within the cloud consumers' specified deadline. Therefore, cloud customers acquire the virtual machines which yield the required performance and produce minimum cost.

In this paper, we propose the Particle Swarm Optimization (PSO) method for cloud provisioning cost optimization. The mechanism of PSO is to allow generated particles (candidate solutions) to move around within the solution spaces. The positions of each particle will be updated at each iterating until the near-optimal position is found. In PSO, the value associated with the position is a real value. In our work, we then designed a decoding scheme to convert the particle position from real values into a discrete solution (integers). Multiple objectives were also used in our optimization framework in order to incorporate multiple independent variables related to cloud purchasing options. The experiment was then conducted with existing workflows mentioned in previous section. The results show that our proposed PSO has a promising performance. The proposed system can be used to decide purchasing options for cloud consumers. The budget can also be estimated for the execution of a large scale scientific workflow.

8. ACKNOWLEDGEMENTS

This work was supported by the Thailand Research Fund through the Royal Golden Jubilee Ph.D. Program (grant no. PHD/0031/2553).

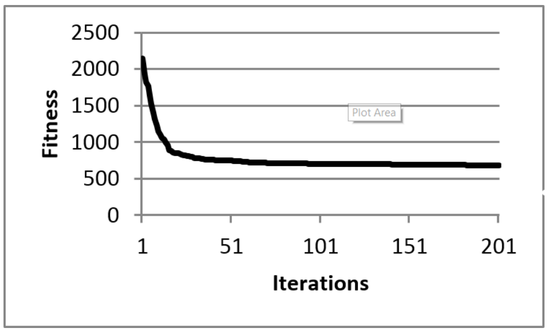

Fig.5: *CyberShake 4,000/0.6.*

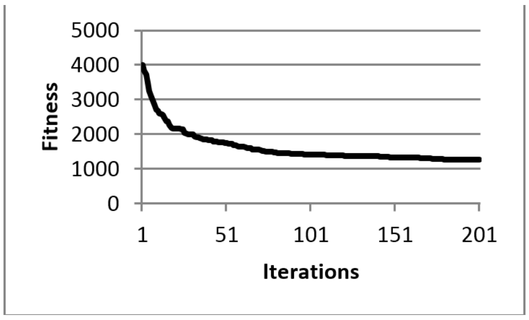

Fig.6: *Inspiral 1,000/2.5.*

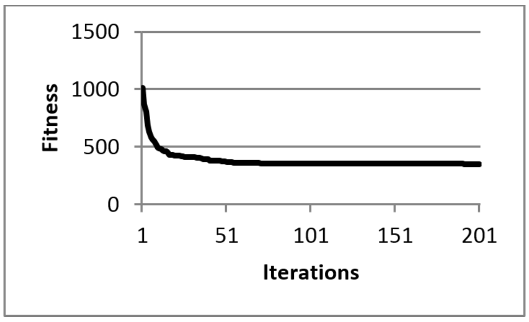

Fig.7: *Montage 6,000/0.2.*

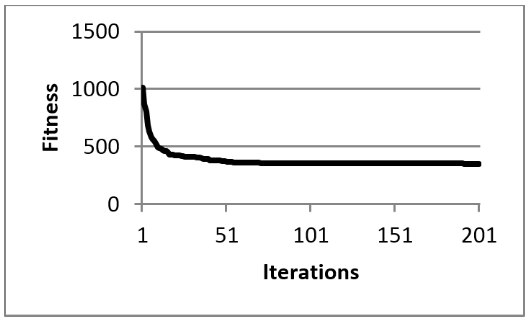

Fig.8: *Sipht 1,000/2.5.*

References

[1] Amazon Elastic Compute Cloud, http://aws.amazon.com/ec2/.

[2] A. Banharnsakun, B. Sirinaovakul, and T. Achalakul, "Job Shop Scheduling with the Best-so-far ABC," *Engineering Applications of Artificial Intelligence*, Vol. 25, No. 3, pp. 583–593, 2012.

[3] A. Banharnsakun, T. Achalakul, and B. Sirinaovakul, "The Best-so-far Selection in Artificial Bee Colony algorithm," *Applied Soft Computing*, Vol. 11, No. 2, pp. 2888–2901, 2011.

[4] S. Bharathi, A. Chervenak, E. Deelman, G. Mehta, M. H. Su, and K. Vahi, "Characterization of Scientific Workflows," *3rd Workshop on Workflows in Support of Large Scale Science (WORKS 08)*, pp.1–10, 2008.

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

[6] E. Byun, Y. Kee, J. Kim, and S. Maeng, "Cost -optimized provisioning of elastic resources for application workflows," *Future Generation Computer Systems*, Vol. 27, No. 8, pp.1011–1026, 2011.

[7] S. Chaisiri, B.-S. Lee, D. Niyato, "Optimization of Resource Provisioning Cost in Cloud Computing," *IEEE Transactions on Services Computing*, Vol. 5, No. 2, pp.164–177, 2012.

[8] R. Chen and C. Wang, "Project scheduling heuristics-based standard PSO for task-resource assignment in heterogeneous grid," *Abstract and Applied Analysis*, 2011.

[9] E. Elbeltagi, T. Hegazy, and D. Grierson, "Comparison among five evolutionary-based optimization algorithms", *Advanced Engineering Informatics*, Vol. 19, No. 1, pp. 43–53, 2005.

[10] A. H. Kashan and B. Karimi, "A discrete particle swarm optimization algorithm for scheduling parallel machines," *Computers & Industrial Engineering*, Vol. 56, No. 1, pp.216–223, 2009.

[11] J. Kenedy, R. Eberhart, and Y. Shi, Swarm Intelligence, Academic Press, 2001.

[12] J. Kennedy and R. Eberhart, "Particle swarm optimization," *IEEE International Conference on Neural Networks*, Vol.4, pp.1942–1948, 1995.

[13] N. Netjinda, B. Sirinaovakul, and T. Achalakul, "Cost Optimization in Cloud Provisioning us-

ing Particle Swarm Optimization," *2012 9th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON)*, pp. 1–4, 2012.

[14] S. Pandey, L. Wu, S.M. Guru, R. Buyya, "A Particle Swarm Optimization-Based Heuristic for Scheduling Workflow Applications in Cloud Computing Environments," *24th IEEE International Conference on Advanced Information Networking and Applications (AINA)*, pp.400–407, 2010.

[15] Y. Shi and R. Eberhart, "A modified particle swarm optimizer," IEEE World Congress on Computational Intelligence, *The 1998 IEEE International Conference on Evolutionary Computation*, pp.69–73, 1998.

[16] Z. Wu, Z. Ni, L. Gu, X. Liu, "A Revised Discrete Particle Swarm Optimization for Cloud Workflow Scheduling," *International Conference on Computational Intelligence and Security (CIS)*, pp.184–188, 2010.

Nuttapong Netjinda received the B.Eng. and M.Eng. in computer engineering from King Mongkuts University of Technology Thonburi, Thailand, in 2005, and 2010 respectively. His research interests include artificial intelligence, swarm intelligence, bio-inspired computation, and optimization model.

Tiranee Achalakul received her B.S. degree in computer engineering from King Mongkuts Institute of Technology Ladkrabang, Thailand, in 1994 and completed her Ph.D. in computer engineering from Syracuse University, New York, in 2000. She is currently an associate professor and is holding an associate dean position at the faculty of engineering, King Mongkuts University of Technology Thonburi, Thailand. Her research interests include parallel and distributed computing, software engineering, Cloud and GPU technology, and data mining application.

Booncharoen Sirinaovakul received the B.Eng. from King Mongkuts Institute of Technology, Ladkrabang, Thailand and the M.S. from the Wichita State University, U.S.A. He received the D.Eng. from King Mongkuts Institute of Technology Ladkrabang, Thailand. He is currently an associate professor in the department of Computer Engineering and also Dean of the faculty of Engineering, King Mongkuts University of Technology Thonburi, Thailand. His research interests are artificial intelligence and natural language processing.