Spring 2014

A TUNABLE WORKFLOW SCHEDULING ALGORITHM BASED ON PARTICLE SWARM OPTIMIZATION FOR CLOUD COMPUTING

Kai Wu
San Jose State University

A TUNABLE WORKFLOW SCHEDULING ALGORITHM BASED ON PARTICLE SWARM OPTIMIZATION FOR CLOUD COMPUTING

A Writing Project

Presented to

The Faculty of the Department of Computer Science

San José State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

By

Kai Wu

April 2014

The Writing Project Committee Approves the Project Titled

A TUNABLE WORKFLOW SCHEDULING ALGORITHM BASED ON PARTICLE SWARM OPTIMIZATION FOR CLOUD COMPUTING

By

Kai Wu

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

SAN JOSÉ STATE UNIVERSITY

APRIL 2014

Dr. Melody Moh	Department of Computer Science
Dr. Teng Moh	Department of Computer Science
Dr. Somik Raha	SmartOrg. INC

ABSTRACT

A TUNABLE WORKFLOW SCHEDULING ALGORITHM BASED ON PARTICLE SWARM OPTIMIZATION FOR CLOUD COMPUTING

By Kai Wu

Cloud computing provides a pool of virtualized computing resources and adopts pay-per-use model. Schedulers for cloud computing make decision on how to allocate tasks of workflow to those virtualized computing resources. In this report, I present a flexible particle swarm optimization (PSO) based scheduling algorithm to minimize both total cost and makespan. Experiment is conducted by varying computation of tasks, number of particles and weight values of cost and makespan in fitness function. The results show that the proposed algorithm achieves both low cost and makespan. In addition, it is adjustable according to different QoS constraints.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLE

1.0 INTRODUCTION

Cloud computing is a emerging technology derived from distributed computing. It provides a pool of abstracted, virtualized resources, including computing power, storage, platforms and software applications over the Internet based on users' demand [1]. Due to its many benefits such as elastic, scalable resource provision and cost-effectiveness, cloud computing has been accepted by more and more users.

Cloud computing offers a great variety of services. Based on the level of services, there are three categories generally. They are Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). IaaS puts servers, storage, networks, and data center fabrics together as demanded by users. Cloud users can then install operating system and deploy their own applications on the cloud. PaaS, on the other hand, provides middleware, database and development tools. It enables users to deploy applications onto a virtualized cloud platform [2]. Finally, in SaaS the complete operating environment, along with applications, management, and user interfaces, are provided to cloud users [3]. Since all cloud services can be access by subscription and run with a pay-per-use model, cloud computing leverages many attractive features to users, including low cost and simple management.

There are many technical challenges faced by cloud providers, such as maintaining high utilization while delivering services that are low cost, short delay, and dynamic deployable. It is critical for cloud providers to maintain an optimal workflow scheduling and management system to meet these challenges.

A workflow is formed by a logical sequence of interdependent tasks decomposed from applications [4]. A cloud workflow system is vital for supporting large-scale e-science and e-business applications [5]. Workflow scheduling component plays a key role in a workflow management system. The scheduler decides which resources will be used, as well as which tasks will be executed on each of these resources, and allocate tasks to the resources. The workflow scheduling problem, like general scheduling problems, is NP-complete. Workflow scheduling algorithms often utilize heuristics and meta-heuristics, including soft computing techniques, to obtain approximated solutions.

In this report, I adopt a workflow scheduling strategy using Particle Swarm Optimization (PSO). PSO, an applied soft computing method developed by Kennedy and Eberhart [6], is one of the most advance evolutionary algorithms driven from nature. PSO approximates an optimal solution by iteratively improving a group of candidate solutions, called particles. Each particle is modified iteratively by the best information from both the individual and the entire group. By collecting the cumulative intelligence of whole group, the group is expected to move toward the most optimal solutions. PSO works well on most global optimal problems [6, 7]. In addition it is simple, effective, and of low computational cost.

Makespan and cost are two key performance measurement criteria assigned by cloud users and considered by workflow schedulers [8-15]. Makespan is the time from the beginning till the completion of the sequence of tasks in a workflow. Different application schedulers may use different policies with different objectives. Some algorithms are designed to achieve minimum cost [9, 12, 14] while others strives for minimum makespan

[13] or for load balance [14]. Most existing algorithms focus on achieving a single optimal criterion [12-14].

In this report, a workflow scheduling strategy to attain a combined minimal cost and minimal makespan is introduced. Moreover, the objective is adjustable between minimal cost and minimal makespan, able to satisfy users' various quality of services (QoS) requirements.

The main contributions of this project are:

1. *A model for a mapping between tasks and resources is formulated, achieving a tunable objective between cost and makespan.*
2. *A PSO-based heuristics is presented to realize the optimal mapping for the tunable objective.*
3. *The heuristics is further improved by addressing bottleneck tasks and thus reduces the makespan even more.*
4. *While most PSO papers simply use a fixed particle number in their experiments, the effect of the number of particles in the PSO performance is studied.*

2.0 Workflow Scheduling Problem

2.1 Scheduling Problem

Scheduling problem is how to allocate tasks with limited resources to achieve some pre-set goals. The "task", "machine", "resources" in scheduling problem are abstract concepts. However, they actually represent extremely wide range of practical objects. During the past few decades, People have done a lot of research on the scheduling problem that has received wide attention from applied mathematics, operations research, engineering and other fields. The linear computation, dynamic programming algorithm and decision analysis in operations research have been widely used to study scheduling problem.

2.2 Scheduling Problem in the Cloud

The goal of cloud computing scheduling is to achieve the optimal scheduling submitted by the user, It should try to improve the overall throughput of cloud computing systems with Specific objectives include the optimal makespan, quality of service,(QoS), load balance, economic principles and so on.

Optimal makespan: Makespan is a very important and common goal in task scheduling. Users usually hope that their tasks can be completed as soon as possible. Optimal makespan is the common goal of both cloud provider and clients.

QoS: Scheduling system must guarantee the QoS specified by the users. On the one side, it needs to improve the efficiency of resources based on application characteristics in order to ensure the efficiency and accuracy of customers. On the other side, it should select and redirect resources dynamically based on users' status changes to meet the user's economy and satisfaction. So the goal is not only to protect users but also helpful for the long-term sustainable development of cloud computing.

Load balancing： Since the number of computers in the cloud computing platform is very large. In additional, the complex composition and different heterogeneous cloud computing platform make load balancing in current could challenging.

Economic principles: Economic is a key factoring in scheduling of cloud computing because of ultra large scale and pay-per-use business model. Market driven cloud users and providers can have mutual benefits from a efficient scheduling system.

Since cloud services require great amount of control and manage resources, a good workflow scheduling is important to manage jobs and tasks. Workflow scheduling plays a key role in the workflow management system[4]. Figure 1 illustrates the cloud workflow scheduling. After submitting workflow by a client, a broker or scheduler is used to run the scheduling algorithm so that the system can start to make decision. In cloud-based infrastructure, the physical machines are virtualized into unified resources called virtual machines (VMs). The scheduler decides which resources (VMs) will be used, as well as which tasks will be executed on each of these resources. It allocates workflow tasks to suitable virtual machine so that the process of computation can be executed to satisfy QoS constraints specified by users such as deadline and cost. This QoS-based optimization aims to minimizing execution cost or make execution time as short as possible and a specified budget.

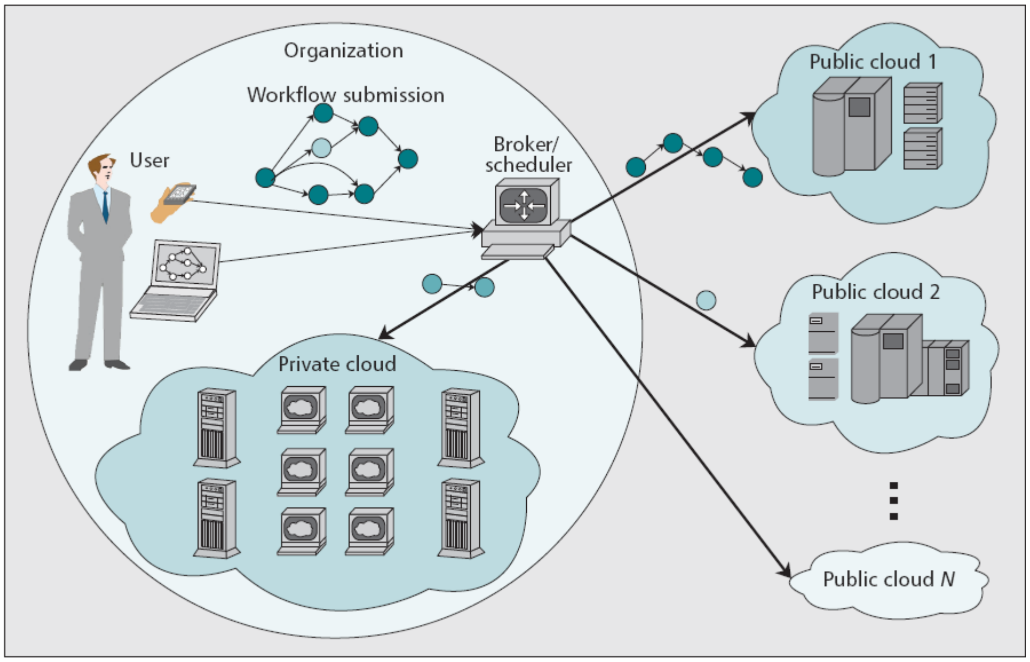

Figure 1. Cloud workflow submission

3.0 Scheduling Problem Formulation

In the following, I adopt the general model and notation used by existing works on PSO-based scheduling [14, 15]. A workflow is usually represented by a Directed Acyclic Graph (DAG), and denoted by $G = (V, E)$. The set of nodes $V = \{T_1, \dots, T_n\}$ represents the tasks in the workflow applications, and n is the total number of tasks in the workflow. The arcs $E = \{d_{ij}\}\ 1 \leq i, j \leq n$ denotes the data dependencies among the tasks. An arc, $d_{ij} = (T_i, T_j) \in E$, implies that T_i transfers data to T_j. In this relationship, T_i is the parent task of T_j, and T_j is the child of T_i. The child task cannot be executed without receiving data transferred from all of its parents. Fig. 1 shows a workflow example of 8 interdependent tasks. Note that any single task can have one or more children (except for the bottom nodes), and any single task can have one or more parents (except for the top node).

Suppose there are a total of m resources in the cloud environment. The resources can be denoted as $R = \{R_1, \dots, R_m\}$. All the resources are interconnected with each other so that they can transfer data among each other. The scheduling problem is to find an optimal mapping M between tasks and resources according to some optimization objective. As mentioned before, cost is a common objective that is more concerned by user; makespan is another objective that is critical for scheduling.

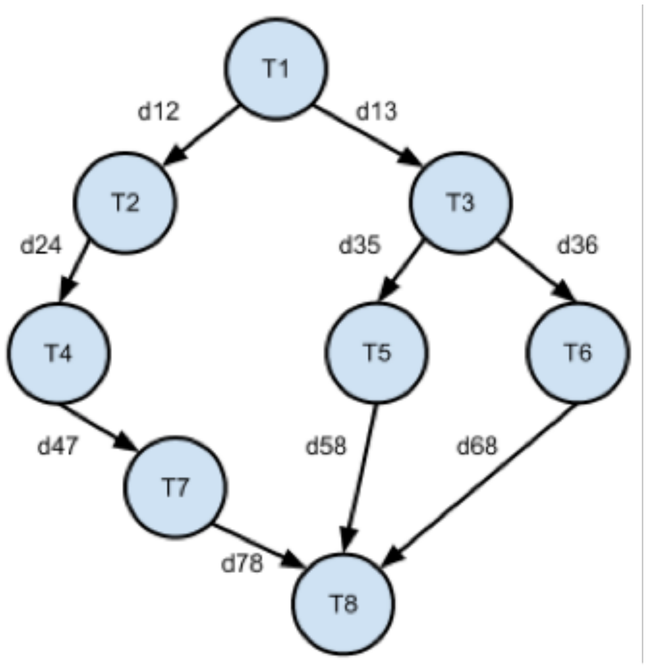

Figure 2. A workflow example with 8 tasks

In the following, I formulate several optimization objectives. Let $Makespan_{total}(M)$ denote the makespan of the workflow with respect to the mapping M:

$$Makespan_{total}(M) = finish\ time\ of\ the\ last\ task - start\ time\ of\ the\ first\ task. \quad (1)$$

The makespan of a workflow is the time duration from the process of first task till finishing all tasks. Since a workflow consists of interdependent tasks, both execution time and transfer time need to be considered.

Next, let $Cost_{exec}(R_i)$ and $Cost_{trans}(R_i)$ be the execution and transfer costs of resource R_i, respectively. $Cost_{total}(R_i)$ denotes the total cost of resource R_i:

$$Cost_{total}(R_i) = Cost_{exec}(R_i) + \ Cost_{trans}(R_i)$$

$$1 \leq i \leq m. \tag{2}$$

Let $Cost_{total}(M)$ denote the total cost of processing workflow *w.r.t* the mapping M:

$$Cost_{total}(M) = \sum_{i=1}^{m} Cost_{total}(R_i). \tag{3}$$

For the objective of minimizing the cost while balancing the load [14], the fitness function is given as:

$$Fitnessfunction_1 = Max(Cost_{total}(R_i)),$$

$$1 \leq i \leq m. \tag{4}$$

The objective is to minimize $Fitness\ function_1$. The reason for not using the total cost of all the resources is to prevent from mapping all the tasks to a single, least-cost resource.

For the objective of optimizing makespan (such as the work by Zhang et al [13]), the fitness function can be defined as:

$$Fitnessfunction_2 = Makespan_{total}(M). \tag{5}$$

The objective is to minimize $Fitness\ function_2$.

In this project I propose an objective of minimizing the *weighted* sum of total cost and makespan; the fitness function can then be defined as:

$$Fitnessfunction_3 = \alpha\, Cost_{total}(M) + (1-\alpha)\, Makespan_{total}(M), \quad 0 \leq \alpha < 1, \quad (6)$$

where α is the weight given to the total cost and $1-\alpha$ is the weight given to makespan. This fitness function can be easily tuned by changing the value α to satisfy the various QoS requirements including budget constraints. Again the objective is to minimize $Fitness\ function_3$.

4.0 Scheduling Algorithm Based on PSO

4.1 Swarm Intelligence

The concept of swarm intelligence stems from observing the behaviors of social groups of organisms bees, ants, geese, fish and other species. Strictly, swarm intelligence is an artificial intelligence model inspired by groups of organisms in nature. These kinds of intelligent patterns need relative number of intelligent individuals to achieve certain types of problem solving capabilities. As intelligent individuals, in generally, without the feedback information from community, the way it is in the solution space travel is . Intelligent individuals can reflect an overall optimization features from entire intelligent group.

Swarm intelligence should follow five rules: Proximity Principle, Quality Principle, Principle of Diverse Response, Stability Principle and Adaptability Principle. Following these five rules does not mean that each individual is quite complicated. The fact is precisely the opposite. The core of swarm intelligence is a group consists of a large amount of simple individuals that can achieve more complex functions through simple cooperation between each other.

Swarm intelligence research has two main algorithms: Ant Colony Algorithm (ACO) and Particle Swarm Optimization (PSO).

In the recent years, ACO algorithms have been successfully applied to a number of discrete and continuous optimization problem such as QAP problems, network routing and scheduling problem. ACO algorithm is relatively mature. Since the PSO algorithm was proposed, it has obtained a lot of attention from domestic and foreign scholars in related fields. The PSO algorithm has a wide range of applying requirements and the development potential.

4.2 PSO Based Scheduling Algorithm

I present a scheduling heuristic optimizing the cost and makespan of workflow using the mapping solution computed by particle swarm optimization based algorithm. PSO is one of the latest evolutionary algorithms inspired by the social behavior of fish schooling or bird flocking [6]. The particle corresponds to an individual bird or fish searching in a natural space. In the PSO algorithm, each particle represents a possible solution. The flock or swarm of particles is randomly generated initially [16]. Each particle has its own position in the space and a fitness value, and has the velocity to determine the speed and direction it flies. A group of candidate solutions (particles) moves around in the search space based on the particles' updated position and velocity so that the PSO algorithm can get a optimized solution.

Particles in the search process update themselves by tracking two best-known positions. One best-known position known as local best position is the individual best-known position in terms of fitness value reached so far by the particle itself. Another best-known position known as global best position is the best position in the entire population. Suppose the number of particles is N. The velocity and position of each particle are calculated by formulations (7) and (8)

$$v_i^{k+1} = \omega v_i^k + c_1 r_1 \left(p_i^k - x_i^k\right) + c_2 r_2 \left(p_g^k - x_i^k\right) \quad 1 \le i \le N, \tag{7}$$

$$x_i^{k+1} = x_i^k + v_i^{k+1} \qquad 1 \le i \le N, \tag{8}$$

$$p_i^k = Best(x_i^k,\, p_i^{k-1}) \qquad 1 \le i \le N, \tag{9}$$

$$p_g^k = Best(p_1^k, \dots, p_N^k), \tag{10}$$

where, the notations in (7-10) are listed in Table 1. Particle's best position is calculated by the fitness function through (9) and (10). The velocity is calculated by three factors each iteration. They are current velocity, local best position and global best position. The position is updated according to its current position and updated velocity. These ensure the particles search around the local and global best positions and converge to a global best position in the limited iteration.

Table 1 Notations used in formulations (7 - 10)

Notation	Description
v_i^{k+1}	Velocity of particle i at iteration $k+1$
v_i^k	Velocity of particle i at iteration k
x_i^{k+1}	Position of particle i at iteration $k+1$
x_i^k	Position of particle i at iteration k
p_i^k	Best position of particle i so far at iteration k
p_g^k	Best position of all particle$s$ so far at iteration k
ω	Inertial weight
c_1, c_2	acceleration coefficients (positive constants)

r_1, r_2	Random numbers in [0,1]

In the workflow scheduling problem, the particle represents a mapping between resource and task. The dimension of a particle is how many tasks the workflow has. In Figure 2, one possible particle for the mapping between 5 resources and 8 tasks is illustrated. The evaluation of each particle is performed based on fitness function. Section 3 gives three fitness functions corresponding to different optimization objectives.

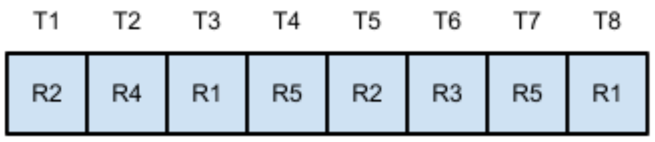

Figure 3. A sample particle.

Table 2 lists the steps of PSO algorithm. The initial step is to initialize each particle's position and velocity randomly. If iteration criterion is not met, the algorithm repeatedly does the following: firstly calculate each particle's fitness value by using the fitness function given in (6), and then update its local best position using (9); calculate global best position using (10); for each particle, update its velocity and position using (7) and (8). Finally, the global best position is the optimal mapping solution.

Table 2 PSO algorithm

```
1:  Initialize particles' position and velocity randomly.
2:  **While** stopping criterion is not satisfied **do**
```

```
3:      For each particle do
4:          Calculate its fitness value using fitness function.
5:          Update its local best position.
6:      End For
7:      Update global best position.
8:      For each particle do
9:          Update its velocity and position.
10:     End For
11: End While
12: Return global best position.
```

After computing the mapping using PSO, the scheduling algorithm dispatches the ready tasks into the resources. Ready task is defined as the one that have received the data transferred from all its parent tasks. Since all of the ready tasks assigned to one specific resource are independent, I then sort the ready tasks in each VM resource dynamically to further improve makespan. Two factors are considered to sort the ready tasks. One is that I schedule the bottleneck tasks with the most descendants first. Another factor is that I set the task that has the shortest execution time to the highest priority and execute this task first.

I enhance the existing PSO algorithm in two respects. First, in the existing PSO algorithm, the fitness function (shown in Eq. 6) for optimizing the mapping between tasks and resources only considers the maximum VM's total cost. This might lead to unevenly distribution of resources and some low cost resources might need to execute much more tasks than the high cost resources. Therefore, the makespan for the total workflow might be much longer. In order to solve this problem, we change the fitness function that considers both makespan and total cost. The total cost of the whole workflow is calculated by Eq. 10. I define the fitness function (shown in Eq. 11) as the sum of weighted total cost and weighted makespan,

$$TotalCost(M) = \sum C_{total}\,(M)_j \quad \forall j \in P \tag{10}$$

$$Minimize(w_1 TotalCost(M) + \; w_2 Makespan(M)) \;\; \forall M \tag{11}$$

where w_1 and w_2 are the weight of cost and makespan. Then I aim to minimize the sum of weighted total cost and weighted makespan. This can ensure algorithm obtain the mapping with both low cost and low makespan.

Secondly, in the existing algorithm after I dispatch the ready tasks into the resources, the tasks are executed in FCFS mode. However, it is not the most efficient way to execute the tasks. All of the ready tasks that are assigned to one specific resource are independent. So I can sort the ready tasks in each VM resource. I consider two factors to sort the ready tasks. One is that I schedule the bottleneck tasks whose descendants are the most first. Another factor is that I set the task with the shortest execution time to the highest priority and execute this task first. The sorting algorithm can be described in Table 3

Table 3 Sorting algorithm

```
1:  For each resource do
```

```
2:  For all ready tasks in resource
3:      Sort tasks in decreasing order of the number of all descendants.
4:  For ready tasks which have the same number of descendants
5:      Sort tasks in increasing order of execution time.
```

5.0 CloudSim Simulation Environment

5.1 CloudSim Overview

CloudSim is a simulation package library developed on the top of SimJava that can run on both Windows and Linux platforms. CloudSim inherits from GridSim programming model so that it can support research and development of cloud computing, and provide the following new features: (1) supporting the modeling and simulation of large-scale cloud computing infrastructure; (2) supporting a self-contained data centers, service agents, scheduling and platform allocation strategy. CloudSim also includes some unique features. Firstly, It provides virtualization engine that is designed to help users to create and manage multiple data centers and collaborative virtualization services; Secondly, at the time of allocating virtualized services and processing cores, it has the flexibility to switch between time-share and space-share. CloudSim platform helps accelerate the development of cloud computing algorithms and methods.

5.2 CloudSim Architecture

In architecture, CloudSim emulator uses a hierarchical organization, which consists of four levels. From the bottom up, they are the SimJava, GridSim, CloudSim, and user code. The bottommost is a discrete event simulation engine SimJava, which is responsible for the implementation of the core functions of high-level simulation framework, such as: query and processing events, build system components (services, clients, data centers, agents and virtual machines), the communication between different components and the analog clock management.

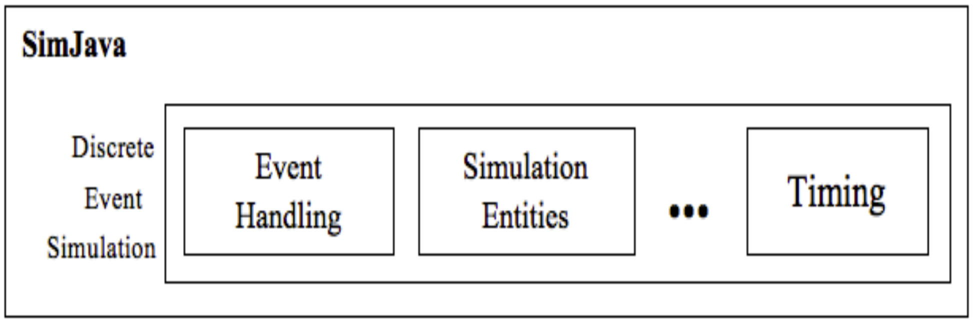

Figure 4. SimJava layer

GridSim is on the top of SimJava, which supports for high-level software components and models a plurality of grid infrastructure including networks and network traffic documents, such as resource-based grid component, data sets, load testing and information services.

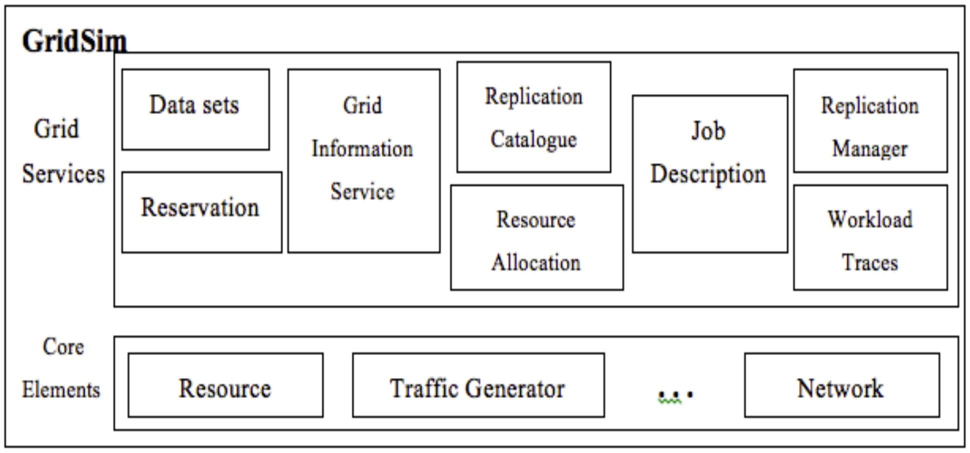

Figure 5. GridSim layer

CloudSim executes in the next layer, which extends the core functionality provided by GridSim. CloudSim layer provides virtual data center management interfaces including virtual machine, RAM, disk storage and bandwidth. It manages core entity (such as VM, clients, data centers, applications) in the simulation process.

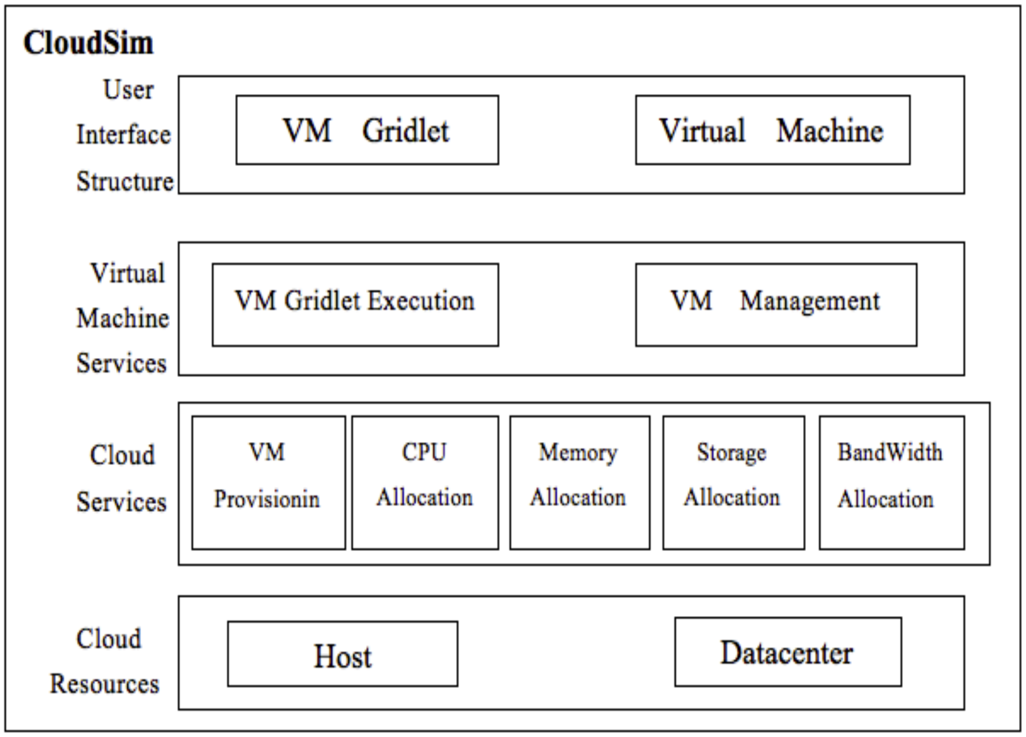

Figure 6. CloudSim Layer

Virtualization layer executes applications on the cloud environment. Virtual machines are running in a client VM and other shared resources. VM management has the ability to define a series of related operations related to VM: to host provides VM, VM is created, VM destruction, VM consolidation.

Cloudlet class represents cloud-based application services such as content distribution, social networks and data center deployments. Each application component has a pre-set instruction length (inherited from the Gridlet components of GridSim), and the amount of data transfers (including prefetching and after taking) to ensure the successful accommodation of this application.

Cloud resource layer: Cloud related core infrastructure services are modeled by the data center, which is used to handle service requests, and requested to perform these services in a VM.

VM Provisioned: assigning VM to the client based on the specific application. This component can help providing VM provisioning policy according to certain optimization rule (user-centric or system-centric).

CPU Allocation: the process of allocating processing cores is accomplished on a client distributor for each client component. Both the number of process cores and allocated computation capacity in each virtual machine will effect this strategy. In additional, Memory Allocation, Storage Allocation, and Bandwidth Allocation have the similar functionalities.

The topmost layer is users' code that is shown in Figure 7. Based on the study of the platform, users can create their own classes, methods, and member variables to achieve specific experiments.

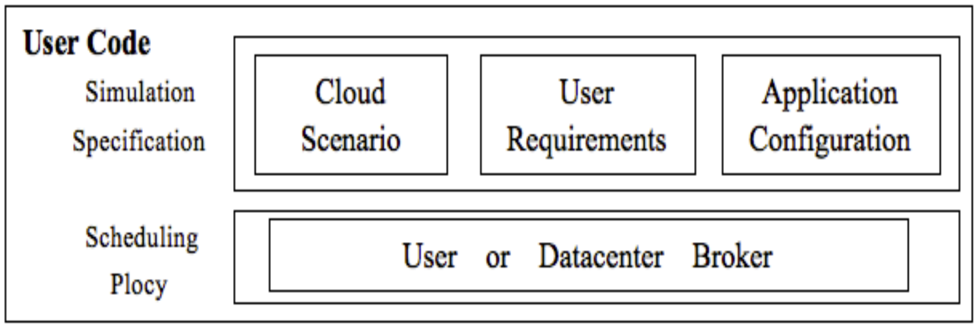

Figure 7. User Code layer

6.0 Implementation and Experiment Result

6.1 PSO Algorithm Implementation

I use the JSwarm[17] package to conduct the simulation in PSO. The PSO class (shown in Figure 8) aggregates Swarm class that also has Particle and FitnessFunction class. Swarm, Particle and FitnessFunction are in JSwarm package. Then I define our own PSOParticle class for PSO scheduling algorithm that extends the Particle class. For FitnessFunction class, I extend it to three different fitness function classes, which are PSOFitnessFunctionCost, PSOFitnessFunctionMakespan, and PSOFitnessFunctionCostAnd-Makespan, respectively. PSOFitnessFunctionCost class only minimizes the maximum value of VM's total cost. PSOFitnessFunctionMakespan class only minimizes the makespan. PSOFitnessFunctionCost-AndMakespan class minimizes the sum of the weighted total cost and weighted makespan, and the weight can be modified by users.

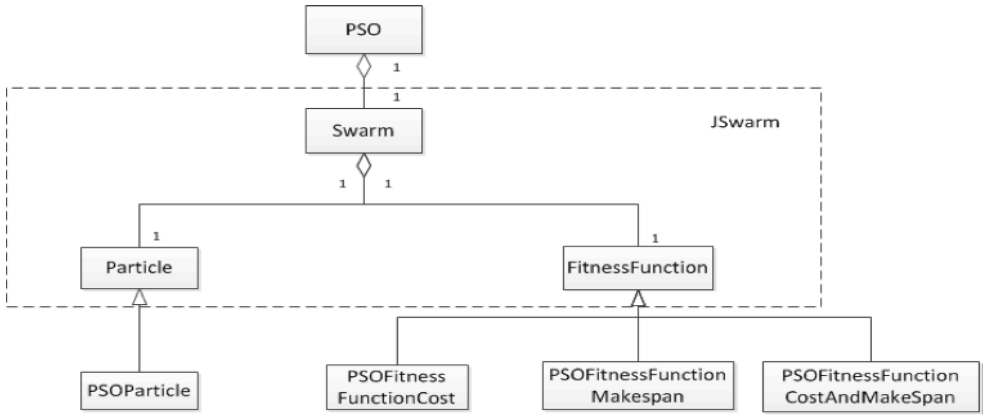

Figure 8. Class diagram of PSO scheduling

6.2 Experiment Setup

For evaluating the performance of the scheduling algorithm, CloudSim 3.0 is used to configure cloud environment and simulate the execution of workflow. The data center that I use in the simulation (shown in Fig. 9) consists of one switch and four hosts each having two VMs. They are constructed by cloud-based interface provided by CloudSim.

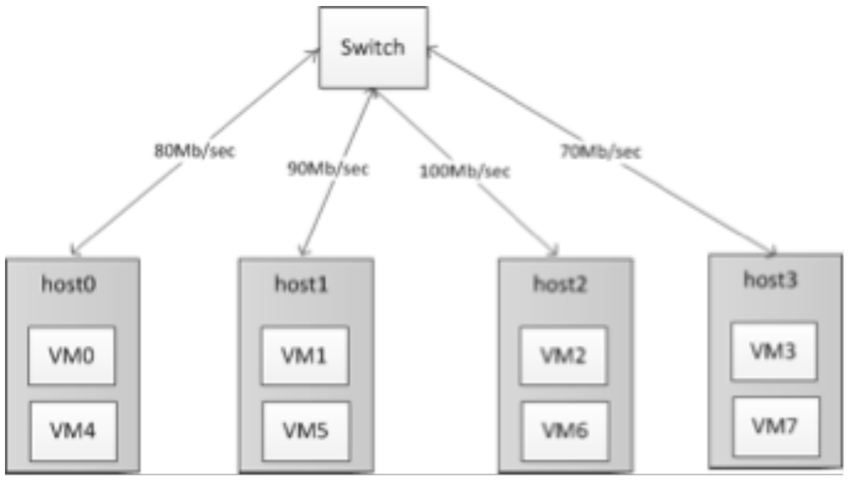

Figure 9. Experimental datacenter infrastructure

The bandwidth of each port of switch is different from each other. The allocation of VMs to hosts uses the default FCFS algorithm in CloudSim. For each VM on the same host, time-shared policy is used so that two VMs in one host can run concurrently. For each task on the same VM, I use space-shared policy so that tasks in one VM are executed sequentially. I modify the inner code of CloudSim to enable sorting ready tasks in each VM. The millions of instructions per second (MIPS) and execution cost of each VM is specified in Table 3, and data transfer cost between different VMs is shown in Table 4. In my cloud service price model, I take Amazon EC2's pricing policy for reference and vary the execution unit price and data transfer unit price based on two essential rules.

Table 4 Transfer cost (cents/MB) between each VM

VMID	0	1	2	3	4	5	6	7
0	0	0.17	0. 20	0.20	0.21	0.21	0.18	0.18
1	0.17	0	0.20	0.20	0.21	0.21	0.18	0.18
2	0.20	0.20	0	0.17	0.22	0.22	0.19	0.19
3	0.20	0.20	0.17	0	0.22	0.22	0.19	0.19
4	0.21	0.21	0.22	0.22	0	0.17	0.20	0.20
5	0.21	0.21	0.22	0.22	0.17	0	0.20	0.20
6	0.18	0.18	0.19	0.19	0.20	0.20	0	0.17
7	0.18	0.18	0.19	0.19	0.20	0.20	0.17	0

Table 5 MIPS and execution cost of each VM

VMID	MIPS	Execution cost (cents/MI)
0	1.011	0.03361
1	1.004	0.03333
2	1.013	0.03444
3	1.000	0.03278
4	0.990	0.03111
5	1.043	0.03528
6	1.023	0.03472
7	0.998	0.03167

Firstly, the execution cost of one task is in proportion to the task's millions of instructions MI and MIPS of VM where the task is executed. Secondly, the data transfer cost is in proportion to the data size and the bandwidth between VMs where the data are transferred. Fig. 9 shows the workflow with 96 tasks used in experiment. Each task has its own MI and there are several data transfers in megabyte (MB) between tasks.

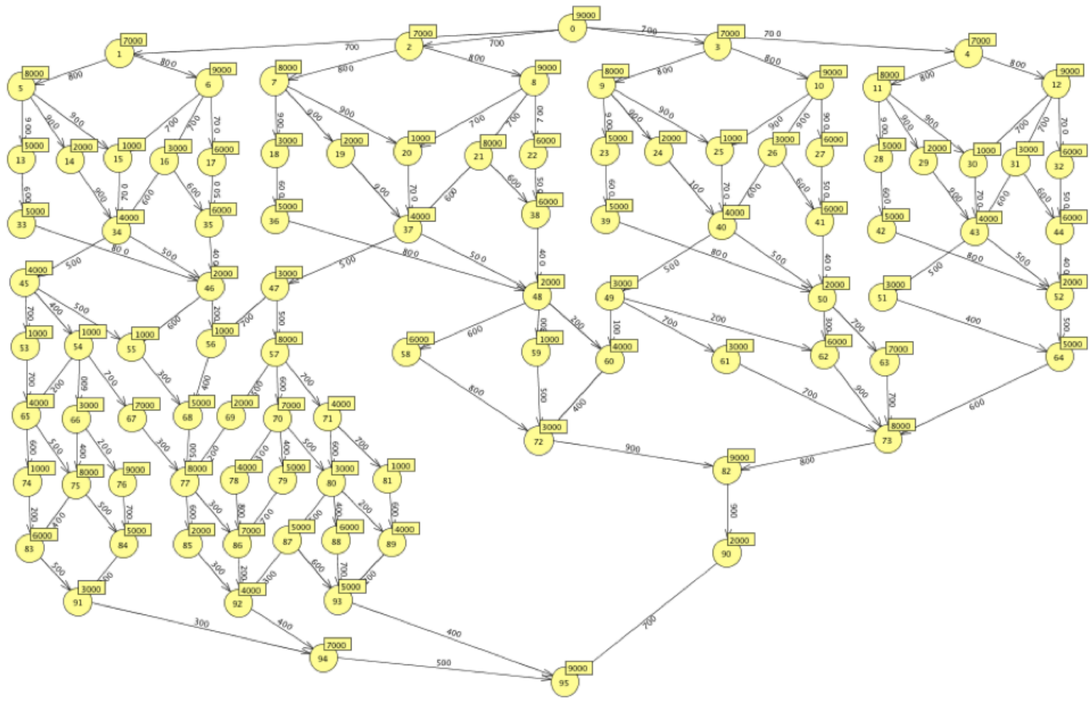

Figure 10. Experimental workflow

In the experiment, I test four algorithms. Algorithm 1, 2, and 3 are the scheduling algorithms using fitness function 1, 2, 3, respectively. The proposed algorithm called Algorithm 4 is Algorithm 3 added with sorting part. Since the makespan and cost are in different order of magnitudes, these two values need to be normalized in fitness function 3. All the experimental results are the average of 30 independent executions. For PSO, the number of iterations is 100.

6.3 Experiment result

6.3.1 The Effect of the Number of Particles

Reference [18] mentioned that the number of particles could influence the resulting performance by a varying amount, depending on the problem being optimized. Therefore, I first conduct the experiments by varying the number of particles. Fig. 10 shows the cost of Algorithm 1 and Algorithm 4 in which the weights of cost and makespan are the same. As the number of particles increases, the cost of both algorithms decreases. The makespan trend of Algorithm 2 and Algorithm 4 is shown in Fig. 11. The makespan of Algorithm 2 decreases along with the increasing number of particles, while makespan of Algorithm 4 fluctuates. From these two figures, I can observe that for Algorithm 1 and Algorithm 2 in which only one parameter is optimized, the more the number of particle is, the better the solution can be found. Whereas there are two parameters optimized in Algorithm 4, and the increased number of particles cannot guarantee that both cost and makespan decrease.

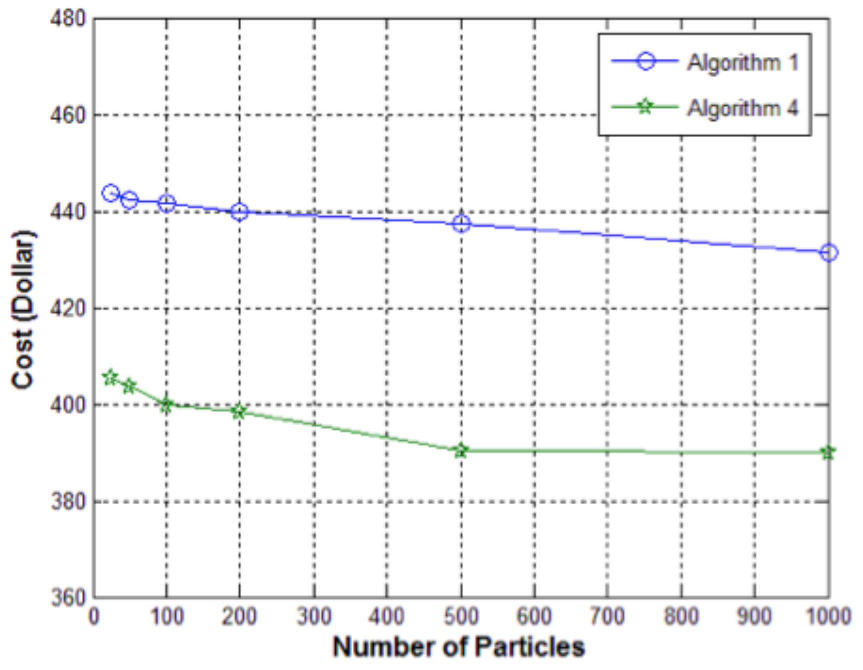

Figure 11. Cost of algorithm 1 and algorithm 4 for different number of particles.

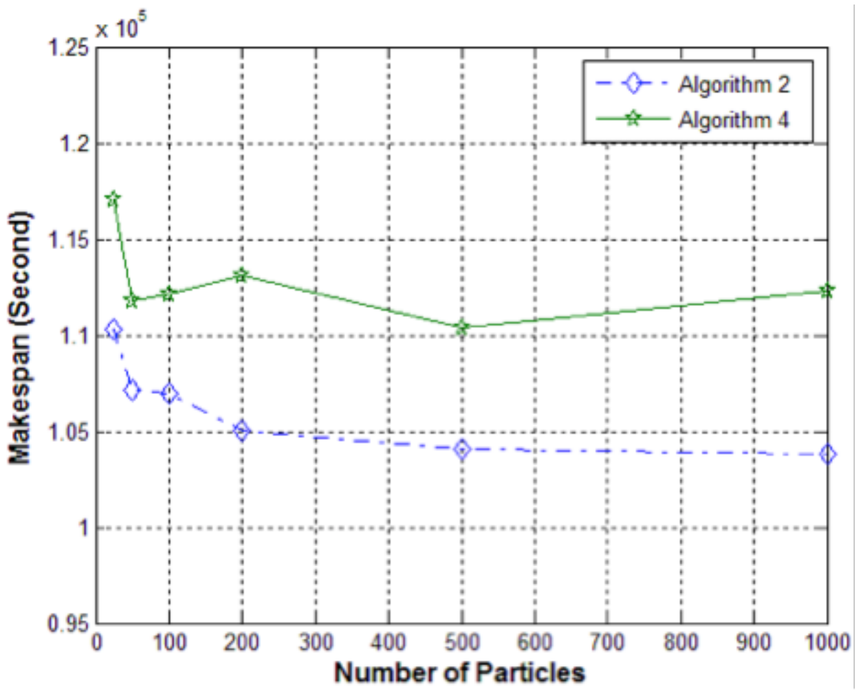

Figure 12. Makespan of algorithm 1 and algorithm 4 for different number of particles.

6.3.2 The Cost Performance

I vary the tasks' MI by multiplying different proportion values and compare cost, makespan and load balance of four algorithms respectively. The numbers of particles in four algorithms are all 500. In Algorithm 3 and 4, the weights of cost and makespan are all 0.5. Fig. 13 shows the comparison of total cost of these four algorithms. I can observe that the cost of Algorithm 3 and Algorithm 4 are very close and are lower than Algorithm 1 and Algorithm 2, while the cost of Algorithm 2 is the highest. This result is reasonable because Algorithm 1 only minimizes the maximum value of resource's cost instead of minimizing the total cost. Therefore, it cannot obtain the lowest total cost. If it minimizes the total cost, then the lowest cost resource might have the most tasks to execute and there is almost no data transfer cost. However this might lead to very large makespan. For the Algorithm 2, it only considers minimizing makespan without considering cost, therefore it has the highest total cost. For Algorithm 3 and Algorithm 4, they consider the total cost instead of the maximum value of resource's cost as in Algorithm 1, therefore they can get the lowest cost. Although Algorithm 4 adds sorting parting to Algorithm 3, it doesn't change the total cost, thus the total cost of these two algorithms are very close.

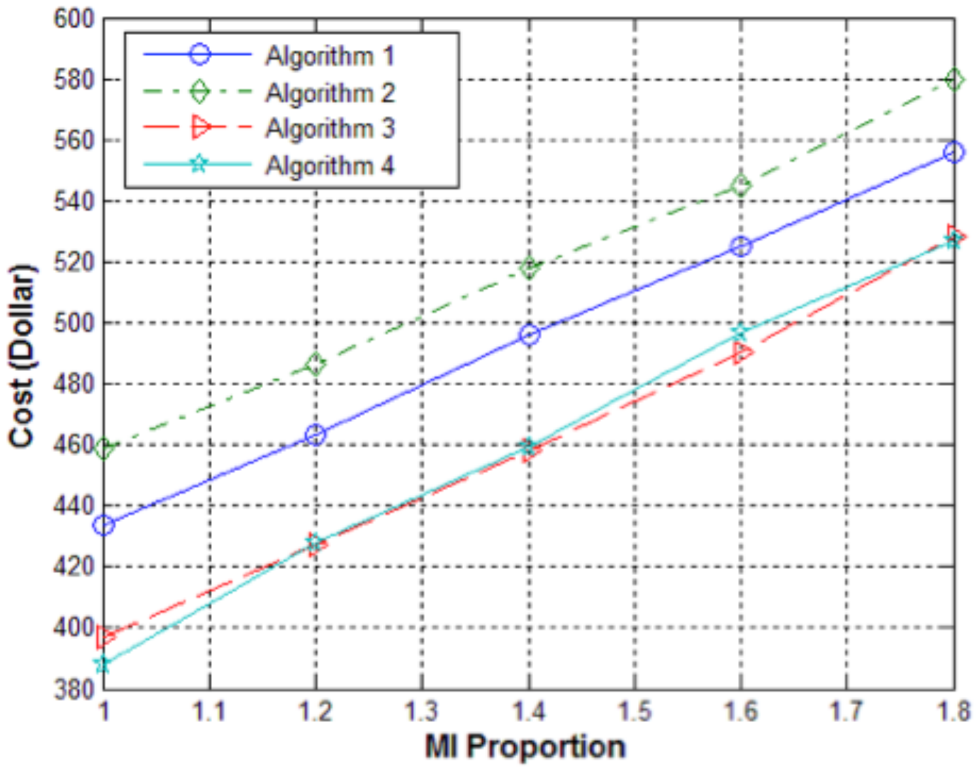

Figure 13. Cost comparison of four algorithms for different MI

6.3.3 The Makespan Performance

The makespan comparison of four algorithms is shown in Fig. 14. I can observe that Algorithm 1 has the highest makespan while Algorithm 2 obtains the lowest makespan because Algorithm 2 is designed to minimize the makespan while Algorithm 1 only takes into account the cost. Since Algorithm 3 and Algorithm 4 both consider the makespan and total cost, they obtain the medium value between Algorithm 1 and Algorithm 2. In addition, Algorithm 4 uses sorting mechanism to further lower the makespan.

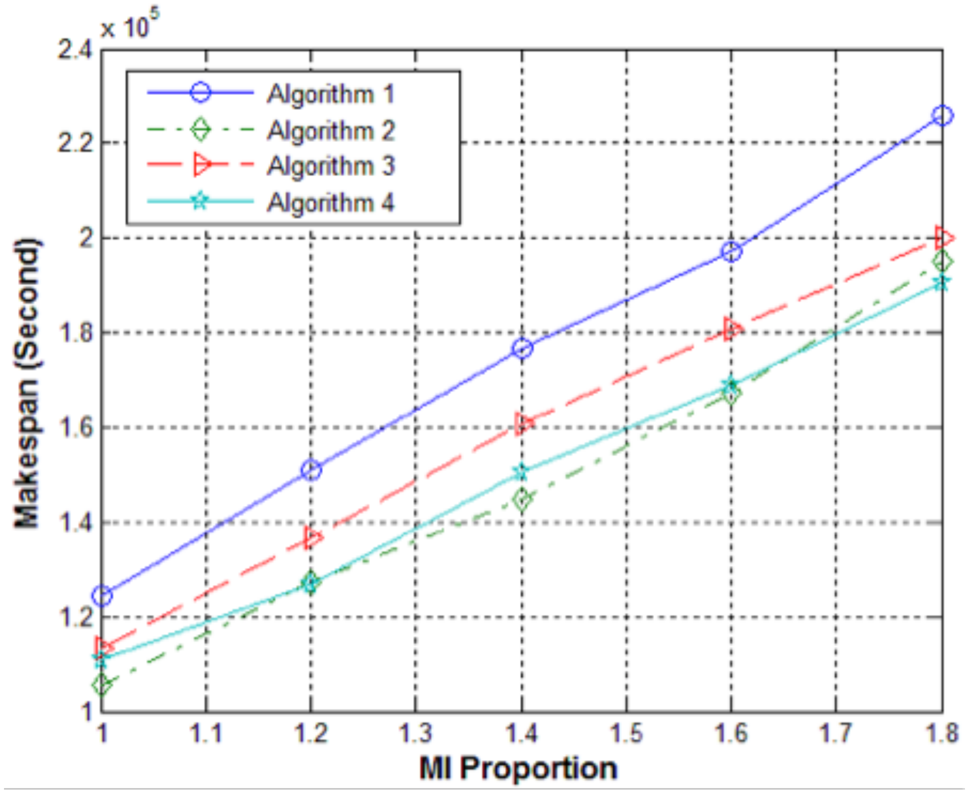

Figure 14. Makespan comparison of four algorithms for different MI

6.3.4 Load Balance Performance

I also compare the load balance of resources of these four algorithms. Fig. 15 shows the average and standard deviation of number of tasks on VMs. The standard deviation is calculated as

$$\sqrt{\frac{\sum_{j=1}^{m}(N_j-\overline{N})^2}{m-1}}, \qquad (11)$$

where m denotes the number of resources (VMs), N_j denotes the number of tasks assigned into the jth VM, and $\overline{N}$ denotes the average number of tasks on VMs. In my experiment, $\overline{N}$ equals to 12. The smaller the standard deviation is, the more average the load distribution of tasks is. I can observe that Algorithm 1 achieves the most balanced load among four algorithms.

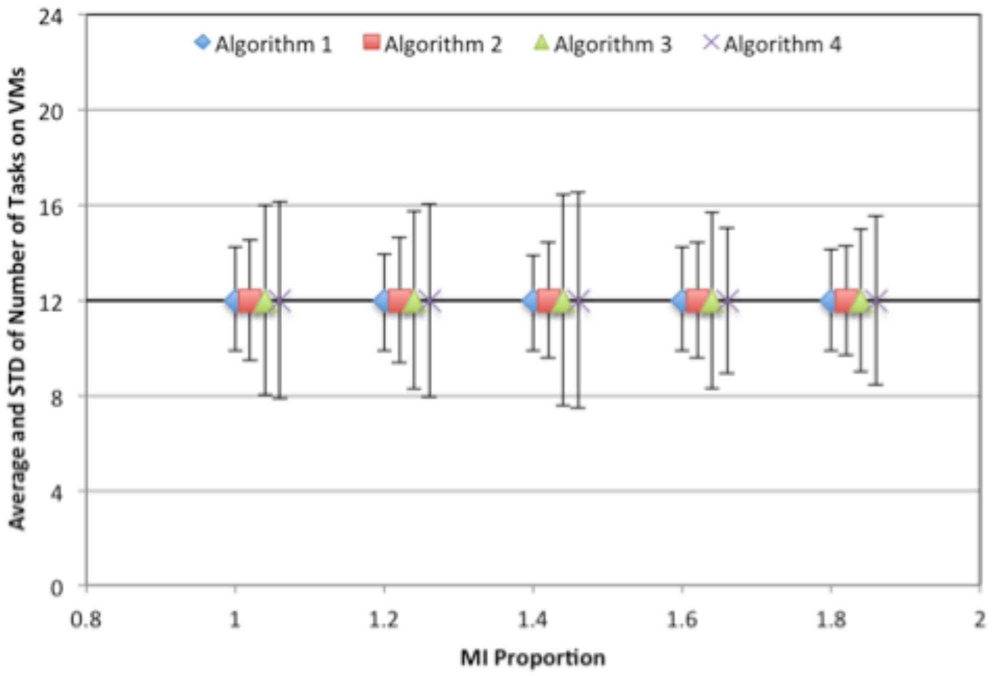

Figure 15. Load balance comparison of four algorithms for different MI

6.3.5 Tuning the Wight Value

In the proposed algorithm 4, the weight values for cost and makespan can be tuned. Fig. 16 and 17 show the cost and makespan results by varying weight values. As the weight placed on optimization of cost increases, the cost decreases while the makespan increases. These results show that my algorithm tunes the cost and makespan well according to different expected weight. Therefore, this algorithm is applicable to different QoS constraints.

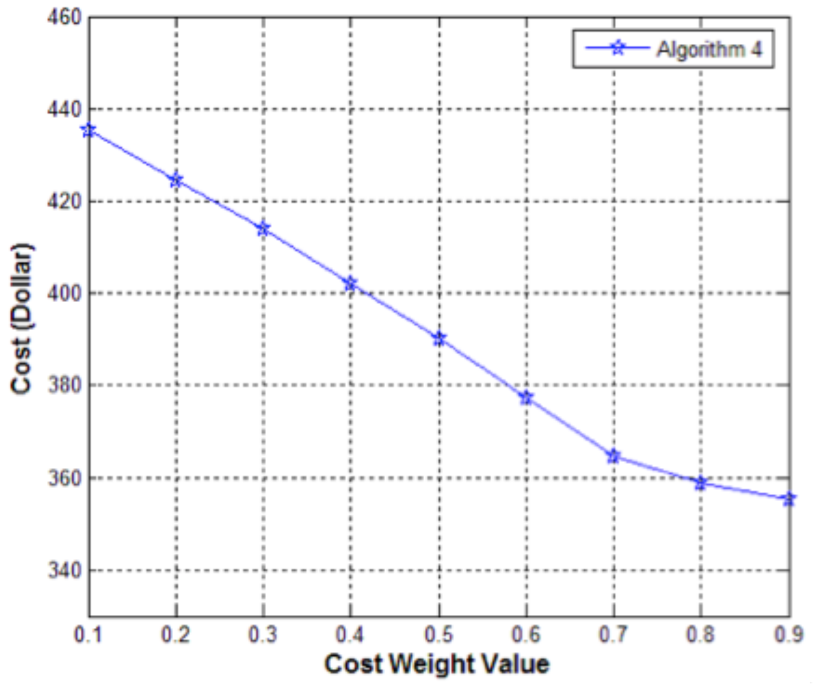

Figure 16. Cost of algorithm 4 for different weight value.

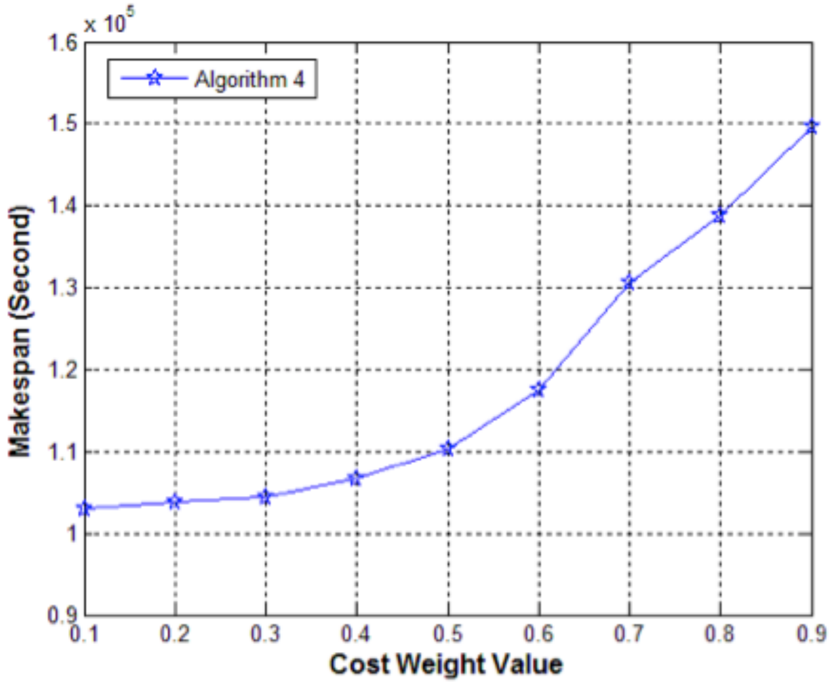

Figure 17. Makespan of algorithm 4 for different weight value.

6.3.6 Transform Makespan to Cost

Different users have different demands for the optimization goal. The weight in the fitness function 3&4 can be tuned to meet their request. However, it is not intuitive and measureable on the first hand. Time is the money. Timing is very important in the business environment. Getting the workflow sooner may represent more users' satisfaction or occupying the market sooner. In the fitness function, we can transfer the makespan(time) into cost(money) by a user specified transform function. I provided two transform function samples. The linear transform function is shown in Figure. 18, which suits for the majority of users. The none-linear function is shown in Figure. 19, which suits for the time sensitive users. They can gain much more profit by executing the workflow quicker.

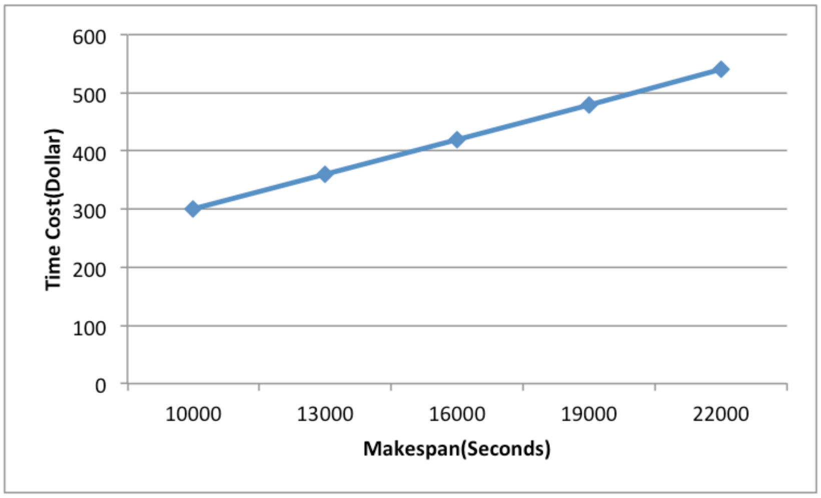

Figure 18. Linear time-cost transform function

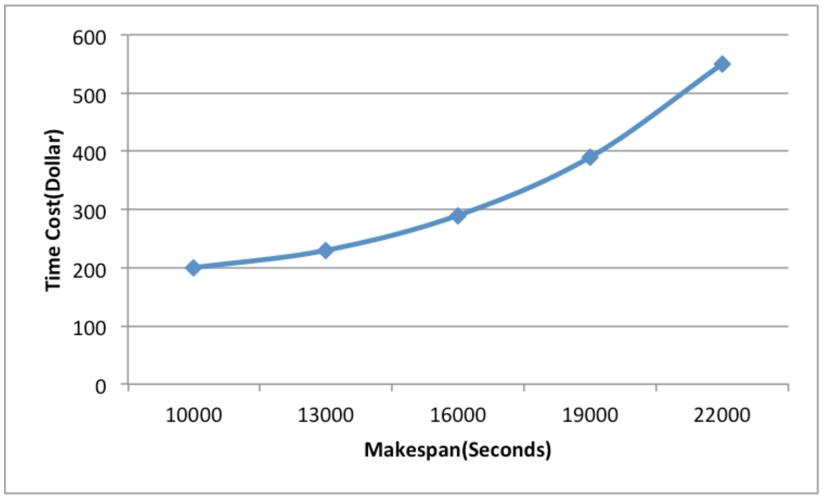

Figure 19. None-linear time-cost transform function

Based on the transform function, the new fitness function is shown in equation. 12

$$Fitnessfunction_4 = \ Cost_{total}(M) + f_{transform}(Makespan_{total}(M))$$

(12)

By using this fitness function, the experiment result is shown in Figure. 20 & 21. The linear time-cost transform function plays more emphasis on the optimizing resource cost. Therefore, The cost result of linear function is lower than the none-linear function. On the makespan side, with the same makespan, the none-linear function has much higher time cost, so it results in a better makespan optimization. In the real use case, each user can specify their own time-cost transform function based on their situation in order to reach a customized optimization goal

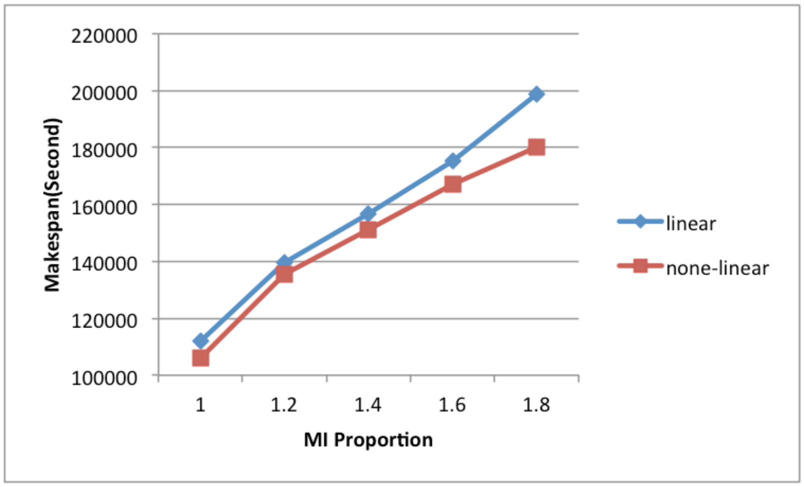

Figure 20. Makespan comparison between linear and none-linear transform function

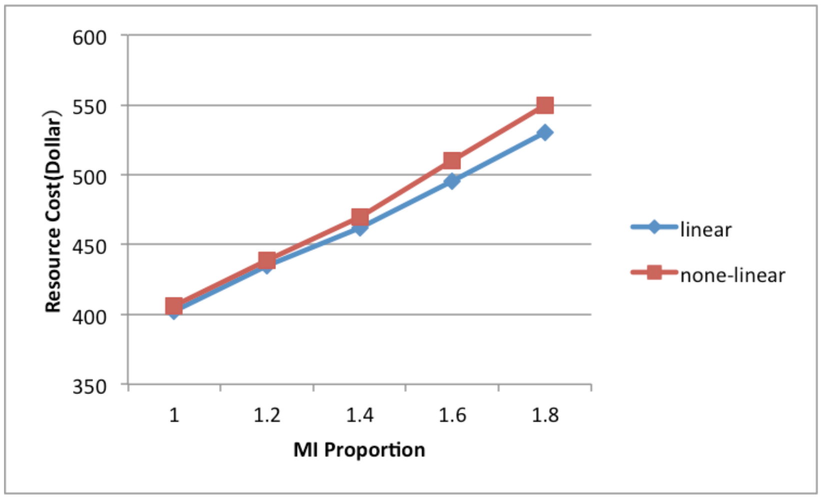

Figure 21. Cost comparison between linear and none-linear transform function

Next, I compare these two algorithms to the four algorithms I proposed previously. We can see the result in Figure. 22 & 23. Using linear transform function can optimize the resource cost as good as algorithm 3 and algorithm 4. On the high workload, using none-linear transform function can optimize the makesapn even better than the algorithm

2. In conclusion, the users can obtain a good optimized result by using a suitable time-cost transform function.

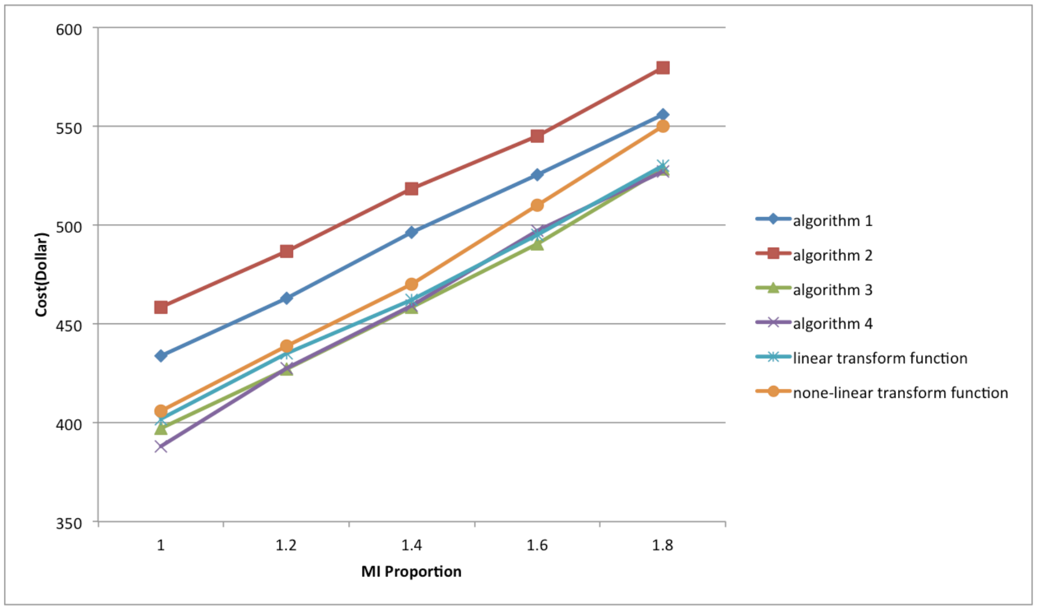

Figure 22. Cost comparison of six fitness functions

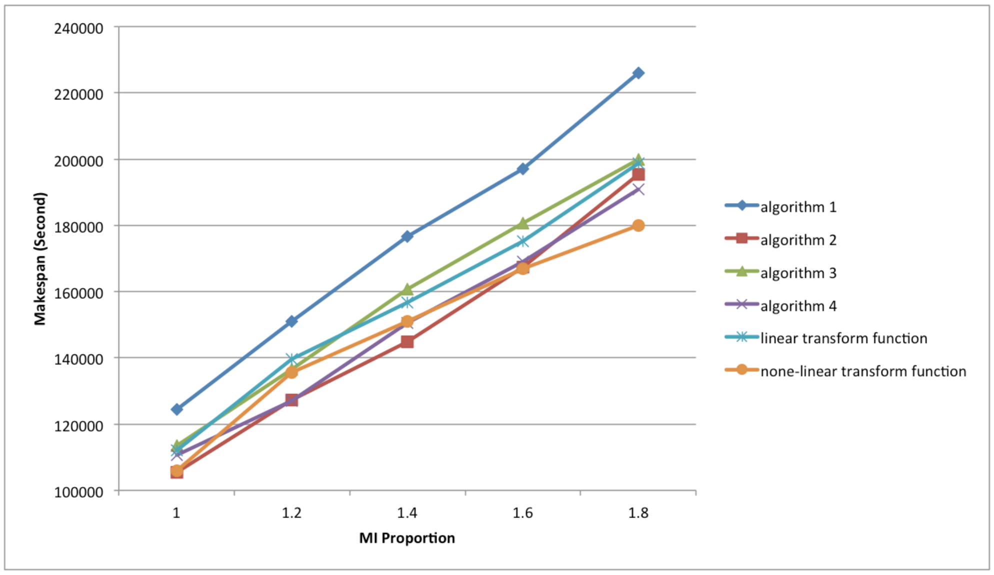

Figure 23. Makespan comparison of six fitness functions

7.0 Complexity Analysis

While Algorithms 3 and 4 perform better than Algorithms 1 and 2 in terms of combined cost and makespan, as shown above, we analyze and compare the time complexity of the four algorithms. Let *N* be the number of particles, *L* be the number of iterations, *n* be the number of tasks, and *e* be the number of edges in the DAG (i.e., the number of data transfers needed among tasks) in the PSO algorithm. The time complexity of the four algorithms is summarized in Table VI. Clearly the four algorithms have comparable complexities. Algorithm 4, which improves over Algorithm 3, does not need a larger time complexity.

Table 6 Algorithm complexity

Algorithm	**1**	**2**	**3**	**4**
Fitness function	$O(n^2)$	$O(e) \leq O(n^2)$	$O(n^2 + e) = O(n^2)$	$O(n^2 + e) = O(n^2)$
For *N* particles	$O(Nn^2)$	$O(Ne) \leq O(Nn^2)$	$O(Nn^2)$	$O(Nn^2)$
For *L* iterations	$\mathbf{O(LNn^2)}$	$\mathbf{O(LNe)} \leq O(LNn^2)$	$\mathbf{O(LNn^2)}$	$O(LNn^2)$
Bottleneck reduction	N.A.	N.A.	N.A.	$O(LNn^2 + n\log n) = \mathbf{O(LNn^2)}$

8.0 Conclusion

The existing PSO scheduling only considers minimizing the maximum value of VM's cost that cannot obtain the lowest cost and makespan. Therefore, I propose the enhanced algorithm which takes into account both total cost and makespan. We implement PSO scheduling algorithm based on three different fitness functions. From experimental results, I conclude that the enhanced algorithm obtains the best result in terms of total cost. For makespan, the result of enhanced algorithm is close to that of the algorithm that only minimizes the makespan. Then I add two sorting strategies to the enhanced algorithm. The experimental results show that by adding sorting part into the algorithm the makespan can be reduced even further.

REFERENCES

[1] I. Foster, Z. Yong, I. Raicu, and S. Lu, “Cloud computing and grid computing 360-degree compared,” Proc. computing environments workshop, pp. 1-10, 2008.

[2] K. Hwang, J. J. Dongarra, and G. C. Fox, Distributed and cloud computing: from parallel processing to the internet of things. Elsevier, Morgan Kaufmann, 2012.

[3] B. Sosinsky, Cloud computing bible. Wiley, 2011.

[4] L. F. Bittencourt, E. R. M. Madeira, and N. L. S. da Fonseca, “Scheduling in hybrid clouds,” IEEE Communications Magazine, vol. 50, no. 9, pp. 48-55, 2012.

[5] Z. Wu, X. Liu, Z. Ni, D. Yuan, and Y. Yang, “A market-oriented hierarchical scheduling strategy in cloud workflow systems,” Journal of Supercomputing, 2011.

[6] J. Kennedy and R. Eberhart, “Particle swarm optimization,” Proc. IEEE International Conference on Neural Networks, vol. 4, pp. 1942-1948, 1995.

[7] D. Bratton and J. Kennedy, "Defining a Standard for Particle Swarm Optimization," Proc. Swarm Intelligence Symposium, pp. 120-127, 2007.

[8] K. Liu, Y. Yang, J. Chen, X. Liu, D. Yuan and H. Jin, “A compromised-time-cost scheduling algorithm in SwinDeW-C for instance-intensive cost-constrained workflows on cloud computing platform,” International Journal of High Performance Computing Applications, vol. 24, no.4, pp. 445-456, 2010.

[9] L. F. Bittencourt and E. R. M. Madeira, “HCOC: a cost optimization algorithm for workflow scheduling in hybrid clouds,” Journal of Internet Services and Applications, vol. 2, no. 3, pp. 207-27, 2011.

[10] M. Xu, L. Cui, H. Wang, and Y. Bi, “A multiple QoS constrained scheduling strategy of multiple workflows for cloud computing,” Proc. IEEE International Symposium on Parallel and Distributed Processing with Applications, pp. 629-634, 2009.

[11] J. Yu and R. Buyya, “Scheduling scientific workflow applications with deadline and budget constraints using genetic algorithms,” Scientific Programming, vol. 14, nos. 3/4, pp. 217-230, 2006.

[12] J. Yu and R. Buyya, “A budget constrained scheduling of workflow applications on utility grids using genetic algorithms”, Proc. Workshop on Workflows in Support of Large-Scale Science, pp. 1-10, 2006.

[13] L. Zhang, Y. Chen, R. Sun, S. Jing, and B. Yang. “A task scheduling algorithm based on pso for grid computing,” International Journal of Computational Intelligence Research, vol. 4, no.1, pp. 37-43, 2008.

[14] S. Pandey, L. Wu, S. M. Guru, R. Buyya, “A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments,” Proc. 24th IEEE International Conference on Advanced Information Networking and Applications (AINA), pp.400-407, 2010.

[15] Z. Wu, Z. Ni, L. Gu, and X. Liu, “A revised discrete particle swarm optimization for cloud workflow scheduling,” Proc. International Conference on Computational Intelligence and Security, pp.184-188, Dec. 2010.

[16] P. Yin, S. Yu, and Y. Wang, “A hybrid particle swarm optimization algorithm for optimal task assignment in distributed systems,” Computer Standards and Interfaces, vol. 28, no. 4, pp. 441-450, 2006.

[17] P. Cingolani. (2005, June 20). JSwarm-PSO [Online]. Available: http://jswarm-pso.sourceforge.net/

[18] D. Bratton and J. Kennedy, "Defining a Standard for Particle Swarm Optimization," Proc. Swarm Intelligence Symposium, pp. 120-127, 2007.