Effective Task Scheduling and Dynamic Resource Optimization based on Heuristic Algorithms in Cloud Computing Environment

Frederic NZanywayingoma[1], Yang Yang[2]
[12] Department of Computer Science and Communication Engineering,
University of Science and Technology Beijing
Beijing, 100083- China
[e-mail: nzanywafre@yahoo.fr]
[e-mail: yyang@ustb.edu.cn]
*Corresponding author: Frederic Nzanywayingoma

Received May 10, 2016; revised November 12, 2016; revised February 28, 2017; revised May 11, 2017; revised July 4, 2017; accepted August 9, 2017; published December 31, 2017

Abstract

Cloud computing system consists of distributed resources in a dynamic and decentralized environment. Therefore, using cloud computing resources efficiently and getting the maximum profits are still challenging problems to the cloud service providers and cloud service users. It is important to provide the efficient scheduling. To schedule cloud resources, numerous heuristic algorithms such as Particle Swarm Optimization (PSO), Genetic Algorithm (GA), Ant Colony Optimization (ACO), Cuckoo Search (CS) algorithms have been adopted. The paper proposes a Modified Particle Swarm Optimization (MPSO) algorithm to solve the above mentioned issues. We first formulate an optimization problem and propose a Modified PSO optimization technique. The performance of MPSO was evaluated against PSO, and GA. Our experimental results show that the proposed MPSO minimizes the task execution time, and maximizes the resource utilization rate.

Keywords: Modified Particle Swarm Optimization (MPSO), Virtual Resource Scheduling, PSO, Global Optimization, Cloud Computing.

The author would like to thank the reviewers for their valuable comments and suggestions that helped us to improve the quality and the correctness of this work. This work was supported by the National Science Foundation of China (Grant Nos. 61202508, 61272432, 61370132,61472033, and 61370092, Fundamental Research Funds for the Central Universities [FRF-TP-14-045A2]).

https://doi.org/10.3837/tiis.2017.12.006 **ISSN : 1976-7277**

1. Introduction

Nowadays, users have realized that the PCs bought few years ago can't move with the development of software; they require a higher speed CPU, a larger capacity hard disk, and a higher performance Operation System (OS). That is the magic of Moore's Law which needs user to upgrade their PCs constantly, to cope with development of techniques[1]. Cloud Computing Services such as Infrastructure as a Service (IaaS), Software as a Service (SaaS), Platform as a Service (PaaS) came to give users the virtual unlimited pay-per-use computing resources (e.g., network resources, servers, and storage resources) with a minimum effort to manage available resources efficiently and gain maximum profit. The PCs that want to access the cloud services just need to have less memory, a light operating System and browser. Users can use the software directly from the cloud, can store data on cloud, can make their applications on cloud platform without installing any software on their machines[1, 2].

According to NIST definition, cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources that can be rapidly provisioned and released with minimal management effort or service provider interaction[3].

Cloud computing has become one of the industry buzz words and a major widely used in various fields of IT world. It is an emerging technology and it processes a large amount of data so that the task scheduling mechanisms work as a vital role. The problem of task scheduling has become a hot topic. At the same time, it is also NP hard problem. Therefore, the main goal of the efficient task schedule algorithm is to decrease total completion time, overall execution time, average response time and to improve the resource utilization rate of the entire system under the condition of meeting the Quality of Service. Cloud computing technology allows users to pay as you need and has the high performance. The rental cost is related to hardware and software, such as the number of CPU frequency, number of core, memory, the size of hard disk and network disk, operating system, databases, network bandwidth, maintenance cost, and location of rented servers, etc[4].

The efficient usage of resources and achieving an optimal allocation of user's tasks are the key issues of task scheduling in cloud computing. Therefore, how to use cloud computing resources efficiently and get the maximum profits becomes the fundamental goals of cloud computing service providers[5]. Scheduling an intensive data or computing an intensive application, it is acknowledged that optimizing the transferring and processing time is crucial to an application program. In our paper, a Task Scheduling Optimization based on Heuristic Algorithms is proposed. Nowadays, cloud computing environment is mostly built according to MapReduce programming model, which is an efficient task scheduling model especially for the generation and processing of large data sets. Therefore, for cloud computing providers, task scheduling and virtual resource allocation with dynamic characteristics is still a challenging problem. In order to find a good method of solving this problem, research is conducted on various heuristic approaches. Thus, key points and difficulties of cloud computing are how to reasonably carry out the task scheduling and virtual resource allocation at the same time.

There are more factors that are affecting the cloud service performance such as the cloud service cost, real-time server load rate, bandwidth utilization especially when users require a higher communication such as in multimedia streaming, reliability of the system, network delay, and task computation complexity. Therefore, these above-mentioned factors yield to

increase total task completion time, overall task execution time, average task response time, and decrease the resource utilization rate of the entire cloud computing system [6].

In this paper, we minimize the cost of the processing by formulating a model for task scheduling and proposing a modified particle swarm optimization algorithm which is based on small position value rule.

The traditional particle swarm optimization algorithms have been integrated into the task scheduling model in the cloud computing environment to improve the quality of service and the fitness function[7]; however, it is common to encounter issues such as local optimization and no interaction between parallel swarms[2].

We resolve these disadvantages of the standard particle swarm optimization (SPSO) algorithm by describing a modified particle swarm optimization (MPSO) algorithm able to identify optimal solutions for the cloud resource scheduling problems. Our contributions to this paper are cited as the followings: (1) we formulate a model for task scheduling in cloud computing to minimize the overall time of executing and transmitting tasks. (2)We design a MPSO algorithm to solve task scheduling based on the proposed mathematical model, compare and analyze with other existing algorithms. We compare the MPSO results with the traditional PSO.

We organized this paper as follows: Section (1) introduces the cloud computing and the basic of PSO optimization. Section (2) shows related work. Section (3) states and formulates the problem. Section (4) presents task scheduling algorithm that uses MPSO and formulate the mathematical model. Section (5) shows the simulation results of the proposed algorithm. Last but not lease, section (6) concludes the paper.

2. Related Work

The most crucial requirement for cloud computing is an effective task scheduling and a virtual resource optimization. Task scheduling is very important to cloud computing resources and it has been a challenging problems. Recently, many researches are being conducted to improve the efficiency of task scheduling and virtual resource utilization. Most of the proposed search algorithms, disregarded the total task completion time, and the resource allocation at the same time. The misuse of the available resource leads to the increasing of server computing time and the user waiting time. Heuristic algorithms have been proposed to solve these complex optimization problems which are mostly non-linear or non-differentiable or combinatorial optimization problems[8].

Due to the complexity nature of optimization problems, constant and linear time-varying values may not work well in many cases. Therefore, using a non-linear coefficient for MPSO could yield to a better performance in some extent. In these days, many heuristic evolutionary optimization algorithms have been proposed to resolve the task scheduling problems [9]. These include Particle Swarm Optimization(PSO), Genetic Algorithm(GA), Deferential Evolution(DE), Ant Colony(AC), Tabu search algorithm, Simulated Annealing Algorithm [10], Artificial bee colony algorithm(ABC) [8, 11, 12], Harmony Search Algorithm[13], and Gravitational Search algorithm(GSA) [14]. The purpose for them is to find the best global optimum among all possible inputs [15].

In [16], Swati Agrawal, consideres a modified PSO to address the problem of premature convergence. Particle Swarm Optimization (PSO) technique suffers from a major drawback of a possible premature convergence. Here, the focus is to free PSO from local optimum solutions; enable it to progress towards the global optimum searching over wider area. For this modified PSO, results have been compared with the base PSO, which is inertia weight PSO.

Numerical experiments done on some benchmark functions compare inertia weight PSO and modified PSO. Also in [17], Said Labed et al., propose a modified hybrid Particle Swarm Optimization (MHPSO) algorithm that combines some principles of Particle Swarm Optimization (PSO) and Crossover operation of the Genetic Algorithm (GA) with two aims: to propose a new hybrid PSO algorithm; to prove the effectiveness of the proposed algorithm in dealing with NP-hard and combinatorial optimization problems. The test results were simulated in matlab7 to assess the efficiency and performance of the MPSO algorithm.
Shafi Ullah Khan et al. [18] propose a MPSO for global optimizations of Inverse problems. In this work, the modified PSO is presented by introducing a mutation mechanism and using dynamic algorithm parameters. The experimental results on different case show that the proposed PSO gets the best results among the tested algorithms. In [18], Ai-Qin Mu et al., use the idea of simulated annealing algorithm to propose a modified algorithm which makes the most optimal particle of every time of iteration evolving continuously. By the testing of three classic testing benchmark functions, it is concluded that the modified PSO algorithm has the better performance of convergence and global searching than the original PSO. The work in [19], takes into consideration both computation cost and data transmission cost. Pandey s. et al., has presented a particle swarm optimization (PSO) based heuristic to schedule applications to cloud resources. In [11] author presents a particle swarm optimization in a multidimensional complex space. Their paper analyzes a particle's trajectory in discrete and in continuous time with the goal of increasing the ability of the swarms in order to find the optimum solution.
The work in [20] used hybrid (PSO and GA) heuristic algorithms to enhance power system stability. Other heuristic algorithms in [21] have been used combined with MapReduce parallel programming model for high performance computing at scale in cloud computing. Their work evaluates system design alternatives and capabilities aware task scheduling for large-scale data processing on accelerator-based distributed systems.
The paper [22] proposed the multiple application co-exist (MACE) method, which minimizes mutual-interference and maximizes resource utilization for resource static allocation, dynamic supplement and resource reserved mechanism [23]. Therefore, we will focus on reducing the processing cost , power consumption, and transmitting time between virtual resources in the cloud computing with the purpose of minimizing the cost and time of the jobs submitted by users. Energy consumption, processing time, and cost. The cost is related to the computing power of the CPU, the memory size, and the bandwidth.
Author [24] suggested a task scheduling using a multi-objective nested Particle Swarm Optimization (TSPSO) to optimize energy and processing time to reduce power consumption and improve the profit of service providers by reducing processing time and author [25] focused on optimizing the energy efficiency in datacenter by using efficient task scheduling to physical servers. To improve the resource allocation in cloud computing, an allocation model using the shortest task completion time and the lowest cost as the constraints to minimize cost and completion time was proposed. Paper [26] proposed a Parallel Bee Colony Optimization Particla Swarm Optimization(PBCOPSO) approach with objective to minimize total execution time and to optimize the resource utilization. The author considered Bee Colony Optimization(BCO) in parallel with PSO to schedule independent tasks. In their experimental results, authors considered to use 4 resources to schedule 40 to 60 tasks. Their developed approach was compared with Min-Min and IBCO (Improved Bee Colony Optimization). Compared to Min-Min algorithm, the proposed new method improved the resource utilization by an average of 5.038% and 3.724% compared with IBCO.
To show the performance of the proposed algorithm (MPSO), our experimental results will be compared against GA and PSO in terms of task scheduling metrics. There exists many

literature on comparison of GA and PSO. PSO and Genetic Algorithm (GA) are two evolutionary heuristics population-based search methods but GA is discrete variables based which is suitable for combinatorial problems. PSO has to be modified to cope with discrete variables. GA and PSO techniques start with a group of a randomly generated population and use a fitness value to evaluate the population. They all update the population and search for the optimum with random techniques. The main difference between the PSO and GA approach is that PSO does not possess genetic operators such as crossover, selection, and mutation. Compared to GAs, the advantages of PSO are that PSO is easy to implement and there are few parameters to adjust [20, 27-29]. Paper [30]compared the results of GA, PSO, ACO, shuffled frog leaping, and memetic algorithms in terms of processing time, convergence speed and quality of the results. PSO was found the best performer than other algorithms in terms of processing time. The elements used for PSO are: Particles, fitness function, local Best, global best, velocity update, position update. And GA employs three operators (crossover, Selection, and Mutation) to propagate its population from one generation to another.
We have explored how PSO/GA work and how they are applied to solve NP-complete problems of task scheduling in cloud computing.

3. Problem Statement

3.1 Assumptions and problem formulation

The task scheduling techniques [31] is to assign incoming tasks to the available resources. According to the scheduling strategies, the poor task scheduling algorithms can significantly affect the efficiency of the whole Cloud Computing system. Let assume that a cloud service provider assigns n tasks to m machines with $(m < n)$. Let $(t_1, t_2, ..., t_n)$ represents the input tasks waiting to be scheduled. Then the task scheduler based on scheduling policy and QoS requirements decides which task to be assigned to which machine from the collection of virtual machines as shown in Fig.3.1. The VMs are represented by $vmid, mips, ram, bw, cpu$. $vmid$ represents VM number, $mips$ represents virtual machines' speed in million instructions per second, ram represents the memory of VM, bw represents the bandwidth, cpu represents the cpu of the VM.
To model our task scheduling problem, we assume that the number of swarm particles correspond with a set of task numbers. The tasks are modeled as a direct acyclic graph (DAG) where G= (V, E). A set of vertices V represents the VMs and tasks and a set of edges E represents the connections between the tasks and VMs and their communication cost. Edge is a pair (v,t) with $v,t \in V$. We denote T_n as the number of tasks and R_m the number of available heterogeneous computing dynamic virtual resources. The objective to model the task scheduling problem is to minimize the completion time and to find the best virtual resource utilization. We evaluate both objective functions using fitness function. Here the fitness of a particle is measured with execution time and communication cost for all tasks.
Let $G = (V, E)$ be a graph with $V = \{t_1, t_2, ..., t_n\}$ as a set of tasks nodes/vertices, and E is a set of edge weights between two tasks t_i and t_k and is denoted the information exchange between these paired tasks. An arc is in the form of $< T_i, T_j > \in E$, where T_{entry} is called the parent task and t_i is the child task. Here, we consider that all tasks are computational and are considered to be dependent to each other accordingly. We consider a complete graph which

means that there is an edge between every pair of vertices. The graph starts with root node and ends with end node. The node with no parent/root node is called an entry node T_{entry} and a node with no child/end node is called an exit node T_{exit} .

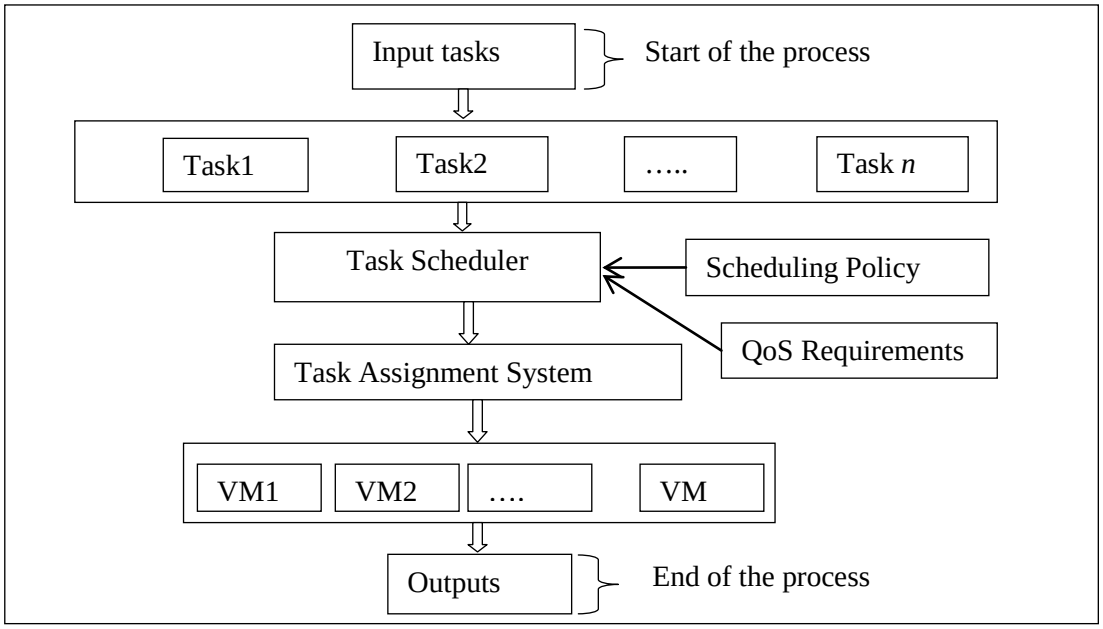

Fig. 3.1 Represents a Task Scheduling model in cloud computing environment[32]

It is assumed that a child task cannot be executed until all of its parent tasks have been completed. **Fig. 3.1** and **Fig. 3.2** illustrate clearly the scenario. Let us consider T_{entry} and T_{exit} as two dummy tasks at the beginning and at the end of the weighted direct acyclic graph (DAG) with zero execution time. We calculate the communication cost $C_{i,j}$ by referencing to the amount of data to be transmitted between resources. The scheduling is considered as a non-preemptive scheduling to mean that there is no interruption when the tasks are processing.

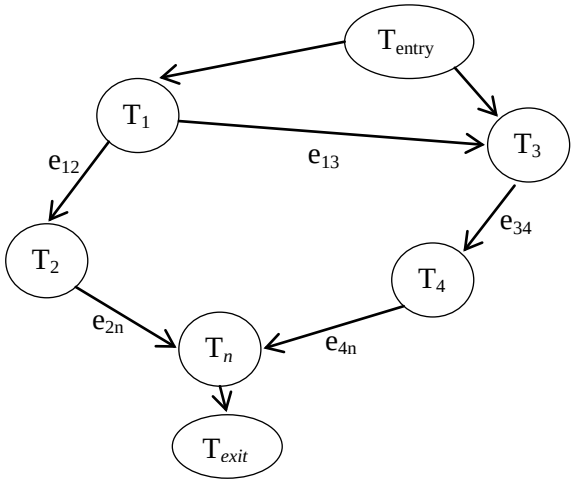

Fig. 3.2. Task graph with n tasks

The mapping of the set of tasks to the available heterogeneous resources in **Fig. 3.3** helps us to compute the maximum completion time of the tasks, minimum execution time, and the execution cost. The cost varies depending on the computing power of the CPU, memory, and bandwidth. To map the above-mentioned set of resources, we consider R_m number of available heterogeneous computing resources, and b_{ij} the bandwidth between resources. Then we calculate the available bandwidth $B = (b_{ij})_{NxN}$ for the available resource. We suppose that there exist a finite number of possible mappings from a collection $T = \{t_1, t_2, ..., t_n\}$ to a collection $R = \{r_1, r_2, ..., r_m\}$ and a large number of pair of tasks and VM resources. The **Fig. 3.3** depicts an illustrative example of the task assignment. Assume that task t_1 is assigned to resource r_1 and r_3, task t_2 is assigned to resource r_2, r_4, and r_j, task t_3 is assigned to resource r_4 and r_j, and t_4 is assigned to r_m respectively.

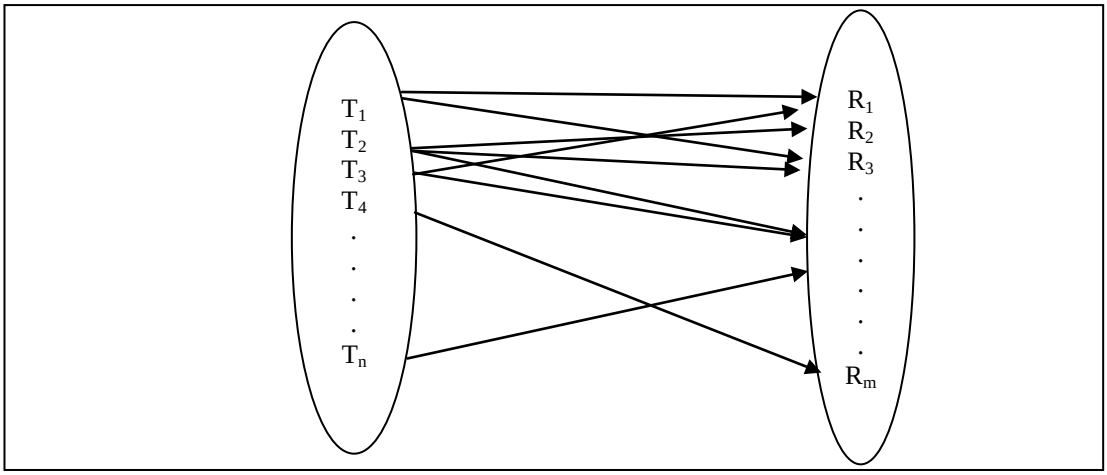

Fig. 3.3. Mapping of the tasks to available resources[33]

We consider a discrete-time model with a collection R_m of machines indexed $1,2,..., m$. Tasks come in with a tagged random mapping number and each task is associated with m number of available resources and they are flocked together according to their indices in an increasingly order into a vector

$$\vec{V} \in \{(r_1, r_2, ..., r_m) \in \{1,2,...,M\}^m \, r_1 < r_2 < ... < r_m\} \quad (1)$$

The distribution relationship between task set T_n and is defined as

$$\chi = \begin{pmatrix} x_{11} & x_{12} & \dots & \dots & x_{1n} \\ x_{21} & x_{22} & \dots & \dots & x_{2n} \\ . & . & . & . & . \\ . & . & . & . & . \\ . & . & . & . & . \\ x_{m1} & x_{m2} & \dots & . & x_{nm} \end{pmatrix}$$ if t_i is running on VM_j then $x_{ij} = 1$ else $x_{ij} = 0$ and $\sum_{i=1}^{n} x_{ij} = 1$ else $\sum_{i=1}^{n} x_{ij} = 0$

x_{ij} represents the position of a particle such that $x_{ij}^{k+1} = \begin{cases} 1 \\ 0 \end{cases}$

To avoid the convergence of the swarm the velocity v_{ij} is limited by $V_{\max}$ and $-V_{\max}$

$X_i = (x_{i1}, x_{i2}, ..., x_{in}),\ i \in [1, m],\ j \in [1, n]$

The weight of a node is $c_{i,j}$ and the computation ability of virtual machine is Ca_j. Each task can be executed on different virtual machines and the execution time of task t_i equals to the ration of the workload and computation ability of the resource r_i

Execution time of the t_i running on VM_j is expressed as $time_{ij} = \frac{MI[i]}{MIPS[j]}$ where $MI[i]$ denotes the lengths of t_i and $MIPS[j]$ is the processing speed of VM_j

The matrix of the execution time is defined as:

$$ET = \begin{pmatrix} ET_{11} & ET_{12} & ..... & ET_{1n} \\ ET_{21} & ET_{22} & ..... & ET_{2n} \\ \\ ET_{m1} & ET_{m2} & ..... & ET_{mn} \end{pmatrix}$$

$T_n = \{t_i \ 1 \le i \le n \ \}$ represents a set of n tasks

$R_m = \{r_j \ 1 \le j \le m \ \}$ represents a set of m resources

ET_{ij} with $1 \le i \le n$, $1 \le j \le m \ \}$ represents a matrix of expected execution times of task t_i on resources r_j

3.2 Objective function formulation

To formulate the objective function, we suppose that user U_i is assigned to virtual resource R_j to submit a set of tasks T_j. The finishing time of T_j can be calculated as the summation of start time and time required executing task T_j: $FT(t_i, r_j) = t(t_i, r_j) + ST(t_i, r_j)$. So the total time spends to complete the user's job by R_j can be defined as $Makespan_j = \max\{FT_j\}$. The objective functions of the new model are expressed as minimizing $makespan_j$ $(j = 1,2,...,m)$. The Scheduler optimizer is implemented by MPSO algorithm to generate an optimal scheduling strategy. The MPSO is an efficient scheduling strategy to face the characteristic of the task cloud computing problem.

4. Methodology

4.1 Particle Swarm Optimization Algorithm overview

PSO was first proposed by Kennedy and Eberhart [34, 35] through simulating of a simplified social behavior model of bird flocking to find food source or fish schooling to protect themselves from a predator. Particle Swarm Optimization (PSO) [3] also known as a heuristic optimizer that optimizes a problem that iteratively tries to improve a candidate solution based on adaptive searching techniques. PSO uses the following parameters: Initial population, swarm, population size, search space, maximum generation. As a powerful optimizer, PSO is applied for uni-processor heterogeneous and preemptive real-time systems. PSO presents the merits of parallel distribution, scalability, easy to realize with high flexibility and strong robustness in dynamic environments. PSO solves many combinatorial optimization problems successfully [5, 15].

PSO puts more emphasis on exploitation than exploration. PSO concentrates on searching around a promising area in order to refine a candidate solution and explores different regions of the search space in order to locate a good optimum. PSO depends on good initial positioning of the particles in the solution space [36]. With their exploitation and exploration, the particle swarms fly through the problem space and have two reasoning capabilities: the memory of their own best position *pBest* and knowledge of the global or neighborhood's best position *gBest* [34, 37]. The same as in cloud computing, each task runs on virtual machine where the resources are distributed virtually like the way particle swarm fly through problem space maintain useful information of their local position and global position. The position of particle is influenced by velocity and has to be updated each time the particle moves from one point to the next position. We assume that the tasks are completely different and are dependent as particles move in swarm and all tasks need to use resources such as CPU, memory, bandwidth, to be executed and they must be measured in terms of cost. The more accurate costs, the more the profits are [38].

PSO has fast speed, but low convergence accuracy. PSO's disadvantages are as follows: the method easily suffers from the partial optimism. PSO uses a number of particles (candidate solutions) which fly around in the search space to find best solution. In PSO population represents the number of particles in the search space. Each particle in PSO should consider the current position $X_i = [x_{i1}, x_{i2},..., x_{iD}]$ vector, the current velocity $V_i = [v_{i1}, v_{i2},..., v_{iD}]$, the

distance to $pBest$, and the distance to $gBest$ to modify its position. Take $pbest_i = \left(x_i^{pbest},...,x_{in}^{pbest}\right)$ and $gbest_i = \left(x_i^{gbest},...,x_{in}^{gbest}\right)$ as the best position of a particle i and its neighbors' best position. Then, the velocity and position of every particle is updated using the two Equations below[39]:

$$v_i^{t+1} = \omega v_i^t + c_1 \times rand_1 \times (pbest_i - x_i^t) + c_2 \times rand_2 \times (gbest - x_i^t) \quad (5)$$

$$x_i^{t+1} = x_i^t + v_i^{t+1} \quad (6)$$

where v_i^t is the velocity of the i^{th} particle at k^{th} iteration, and ω is a weighting function. c_1 and c_2 are the weighting factor which increase the performance of PSO, $rand_1$ and $rand_2$ are the random numbers between 0 and 1which give the PSO a more randomized search ability; x_i^t is the current position of the i^{th} particle at the k^{th} iteration; $pBest_i$ is the variable to store the best solution obtained by the i^{th} particle; $gBest_i$ represents the particle position or global position as well. To achieve a high performance, we set the inertia weight as

$$\omega = \omega_{\max} + \frac{(\omega_{\min} - \omega_{\max})}{iter_{\max}} \times iter \quad (7)$$

$\omega_{\min}$ and $\omega_{\max}$ are the starting and ending inertia weight which are responsible to control the PSO algorithm's stability. Their best values are between 0.2 and 0.9 respectively. $iter$ and $iter_{\max}$ represent the current and maximum iterative time which we set to 1000. ω is set to 0.4. The maximum number of inertia weight is assigned to particles with fitness values greater than average fitness value while minimum value of inertia weight is assigned to particles with fitness values lesser than fitness average value. The first part of the above-mentioned formula, ωv_i^t , provides exploration ability for PSO. The second and third parts, $c_1 \times rand \times (pbest_i - x_i^t)$ and $c_2 \times rand \times (gbest - x_i^t)$, represent private thinking and collaboration of particles respectively. The inertia weight ω balances the global optimization capability and the local optimization capability.

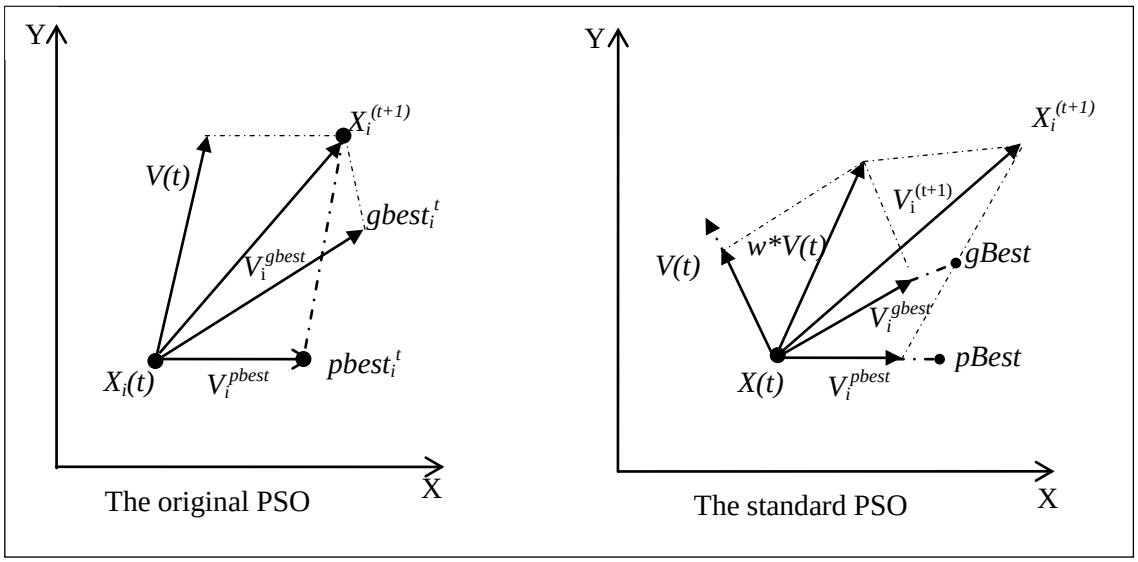

Fig. 4.1. The movement of particles: a concept of modification of a searching point by PSO

4.2 The implementation of MPSO Algorithm for Task Scheduling and Virtual Resource Allocation in Cloud Computing

Cloud computing technology brings the computing resources and storage resources in different geographical positions into a resource pool through virtual technology. The users have to use them and then need to release them so that they can be reused. In this way, we can calculate the average computation cost and computation time of all tasks on all the resources. The MPSO can be used to solve this non-linear complex optimization problem due to its advantages of being less complex operations and parameters. MPSO presents a good deal in cloud computing because cloud computing server cluster can fast realize resource discovery, resource matching, scheduling production, and task average running time [5].

The proposed MPSO enlarges the scope of excellent positions and enhances global search ability in order to improve the performance of particle swarms. In this work, MPSO adjusts the value of inertia weight ω, particle velocity, and updates of particle position. The inertia weight ω balances the global optimization capability and the local optimization capability.

During the resource discovery, and scheduling process, the cloud system uses *pBest* which is the best location the particle has achieved so far. *pBest* can be viewed as the particle's memory and does not depend only on the value of fitness function. It also depends on other constraints. *gBest* is the best position that neighbors of a particle have achieved so far. *gBest* takes the whole population as the neighbors of each particle.

The selection of *gBest* consists of three steps: determine the neighborhood, select the *gBest* among the neighbors, and compare fitness values among neighbors. In this MPSO algorithm, most of the steps are the same as PSO algorithm but ours aims at enhancing the global search ability, swarm diversity by increasing the chance to find a better solutions and exploitation ability. In first steps, all the particles are initialized. The fitness value of all particles is calculated. The *pBest* and *gBest* are calculated. The process of the MPSO algorithm is shown in the **Fig. 4.2** below:

As we can see from the **Fig. 4.2** above, the heuristic algorithms are essential to solve real-world and complex problems. PSO and our proposed MPSO are close to each other structurally but the movement of particles in neighborhood represents a solution for the problem. The particle moves by the direction on the *pBest* and *gBest* until reaching the maximal number of iteration. The algorithms are described as follows:

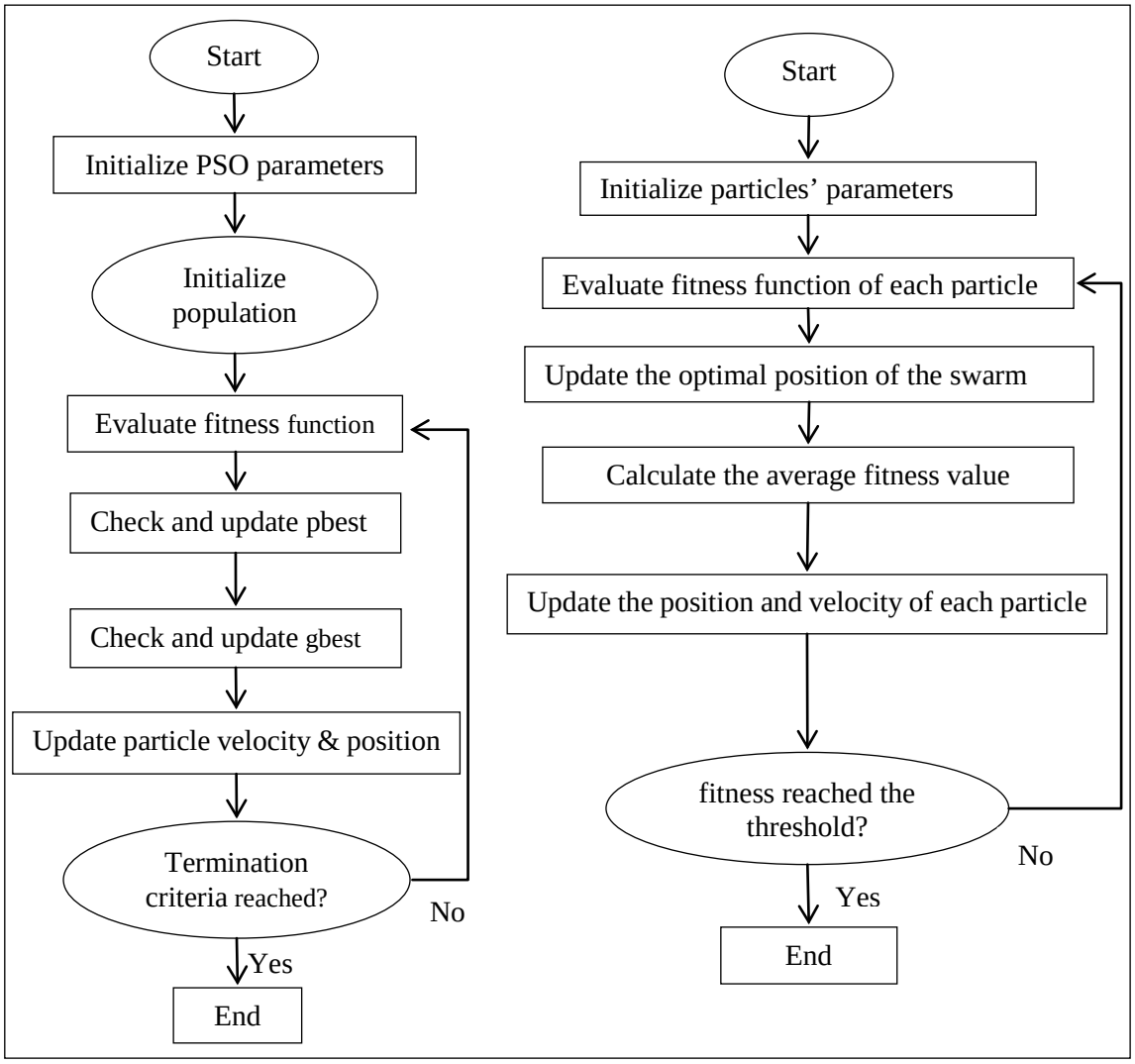

Fig. 4.2. Flow chart of Standard PSO and MPSO

Algorithm1–Pseudo code of Particle Swarm Optimization (PSO) algorithm

1: *Initialization:* Start initializing particles with random positions
2: *Conversion:* Convert the continuous position vector to discrete vector.
3: *Fitness:* Calculate fitness value for each particle using fitness function
If current fitness is greater than fitness *pBest* , then
4: *Calculating pBest* : Calculate the best particle *pBest* and assign it's best position value to *gBest*
5: Calculate velocity for each particle
6: U*pdating:* Update the particle velocity and the swarm best known position until reaching the termination condition based on formula *(6)* and *(7):*

$$v_i^{t+1} = \omega v_i^t + c_1 \times rand_1 \times (pbest_i - x_i^t) + c_2 \times rand_2 \times (gbest - x_i^t)$$

$$x_i^{t+1} = x_i^t + v_i^{t+1}$$

Where

ω **=inertia**

c_1, c_2 =uniformly distributed random numbers

$pBest$ =*best position of each particle*

$gBest$ =best position of the entire particles in a population

t = iteration

7: Repeat 2 to 6 until the stopping condition is satisfied (reaching maximum number of iterations or when no change in fitness value for a consecutive iteration)

8: *Output:* Print the final solution as the best particle

Algorithm2 – Pseudo code of modified Particle Swarm Optimization(MPSO) algorithm

1: Get the best solution of the particles

2: Set particle dimension as equal to the size of available tasks

3: Randomly, initialize particles position and velocity of each particle;

4: Calculate the best fitness value of each particle, concern the particle with the better fitness value, and compare its fitness value with the fitness of its $pBest$;

if $F(X_i(t)) < F(pBest_i)$

$F(pBest_i) = F(X_i(t))$

$pBest_i = X_i(t)$

5: Calculate the best particle as $gBest$ with the best fitness value,

if $F(X_i(t)) < F(gBest_i)$

$F(gBest_i) = F(X_i(t))$

$gBest_i = X_i(t)$

6: Calculate velocity and update their positions

MPSOupdatevelocity() //change the velocity of the particle according to (5)

MPSOupdateposition() //change the position of the particle according to (6)

7: Check if the stopping condition or the maximum iteration is reached. otherwise, repeat from step3 until a stopping criterion is satisfied.

Algorithm3- MPSO-Based Virtual Resource Scheduling

1: Set the virtual resource parameters and the weight of the nodes and edges from the DAG;

2: Select tasks to be allocate to the available resources $r_j \in R$ according to their priorities p_j

3: For every $r_j \in R$, do

4: Schedule all tasks from r_j

6: End

7: End

8: For the nodes $r_j \in R$

9: The task is assigned to a virtual resource r_j for execution based on the MPSO algorithm

10: End

4.2 Algorithms Descriptions

In this section, the details of the proposed MPSO algorithms are explained clearly. The above task scheduling algorithms in the cloud-computing environment are described based on PSO elements as follows:

The particles are defined as the available real tasks t_n; the fitness function is defined as the function used to find the optimal solution; local best is defined as the best position of a particle among its all positions visited so far; global best is defined as the position where the fitness is achieved among all the particles visited so far; Inertia Weight is defined as the value used to balance the exploration-exploitation trade off; the velocity update is defined as a vector to calculate the speed and direction of the particle; and then the position update is defined as a global optimal position of a particle.

The PSO parameter settings are as follows:

Initial population is a set of particles at a starting time and are generated randomly; the swarms are disorganized of moving particles that tend to cluster together while each particle seems to be moving in a random direction; population size is the number of particles which can be fixed accordingly; search space is the range in which the algorithm computes the optimal control variables. We set the lower bound and upper boundary to 0 and 1; maximum generations are the maximum number of generations allowed for the fitness to converge with the optimal solution. From the initialization stage, the algorithm sets the number of particles, initializes the particle position vector and velocity vector of each particle in the particle swarm search space, where in $x_r \in [l_r, u_r], |v_r| \le v_r^{\max}$

The maximum speed of particle in *n*-dimensional has to be set in order to limit the velocity of a particle. $v_r^{\max} = \beta \times (\frac{u_r - l_r}{u_r + l_r})$ (8)

where $\beta \in [0,1]$, and v_r is the velocity of the n^{th} dimension particles, u_r and l_r are the upper bound and lower bound of the *r*-dimensional search space. The speed limit can cause the convergence of the particle to the global optimal position. The second step calculates the inertia weight ω and updates the velocity of the particle. If $v_r \ge v_r^{\max}$, then $v_r = v_r^{\max}$, if $v < -v_r^{\max}$, then $v_r = -v_r^{\max}$. Update the particle position from the initial position up to the next position. If $x_r < l_r$, then set $x_r = l_r$, if $x_r > u_r$, set $x_r = u_r$. The algorithm calculates the fitness of each particle and updates the *pBest* and *gBest* optimal of each particle position in the particle swarm. When it reaches to the maximum iteration or finds the ideal result, it will close up the process otherwise it will repeat the process from step2 to update the velocity of each particle until the stopping criterion is satisfied.

Let us assume that we know the size of input and size of output of each tasks and assigned as edge weight e_{k_1}, e_{k_2} in **Fig. 3.2**. Depending on the number of tasks completed, the ready list is updated, which will now contain the tasks which parents have completed execution. It will update the average values for communication between resources according to the current bandwidth. When the remote resource management systems are not able to assign task to resources according to our mappings strategy due to resource unavailability, the computation of MPSO makes the heuristic dynamically balanced to other tasks' mapping.

Therefore, in this problem, the particles are the tasks to be assigned and the dimensions of the particles are the number of tasks in a cloud system. The value assigned to each dimension of

particles is the computing resource to a task. The performance of each particle is evaluated by fitness function described above where i represents the number of particles and j represents the virtual resource node number. The particle calculates their velocity and updates their position accordingly. The evaluation step is carried out until the specified number of iterations is reached. The performance of MPSO varies according to the variation of the computing resource cost. This variation depends on the way cloud service varies pricing policies depending on the type and capabilities of the virtual resources.

5. Simulation and Analysis of the Results

5.1 Simulation Environment

In order to prove the feasibility and to show the performance of the proposed algorithm, we chose to use a comparison method of MPSO algorithms against GA, and standard PSO. We take into consideration various parameters like number of tasks, number of CPU, number of VMs, number of iterations, swarm size, population size, as shown in Table5.1 below. The simulation was conducted into two scenarios: 1) Using the Matlab program running on Intel(R) dual-Core(TM)i5-4590 CPU@3.30GHz, with 6.00GB of memory and 2) Using CloudSim toolkit running on Windows 7 operating system with Intel(R) dual-Core(TM) i5-4590 CPU@3.30GHz, with 6.00GB installed memory. We considered using a maximum of 10 virtual machines.

Table 5.1. Parameters used for particle swarm optimization and genetic algorithm

PSO parameters	Population size	40
	ω_{max} ω_{min}	0.9 0.1
	C_1	2.0
	C_2	1.0
	Number of iterations	1000
GA parameters	Population size	40
	Crossover probability	0.7
	Mutation probability	0.01
	Number of iterations	1000

The reason we have chosen these parameters is that c_1 and c_2 are the weighting factors that increase the performance of MPSO. The best range for c_1 is 1.5 to 2 and the best range for c_2 is 1 to 2.0. We set the inertia weight ω_{min} and ω_{max} as the starting and ending inertia weight which are responsible to control the MPSO algorithm's stability. Their good range values lay between 0.1 and 0.9. $iter_{min}$ and $iter_{max}$ represent the current and maximum iterative time which we set to 1000. The population size is fixed at 100 particles but it can be any number. For purpose of comparing PSO and MPSO, our population size is fixed at 100 particles. Our research has found out that fixing the values as they are represented in Table5.1 provides the best convergence rate for all test problems considered.

Scenario 1: Iteration of the particle position and velocity:
In each PSO iteration step, each particle moves from one position to the next position based on its velocity; by moving it reaches to different prospective of the problem. The basic particular equation was represented in (5) and (6). Each particle was first evaluated to find the particle

objective function value. The result is revealed in **Fig. 5.1** below.

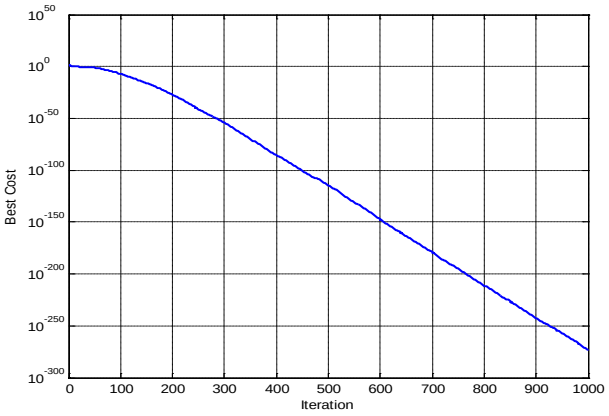

Fig. 5.1. MPSO representation of the best cost at 1000 iterations

The **Fig. 5.1** simulates the best cost for the proposed MPSO algorithm. The cost varies gradualy due to the increase of number of iteration. Increased number of iteration increases the quality of solution which lead to the good cloud resource solution

.

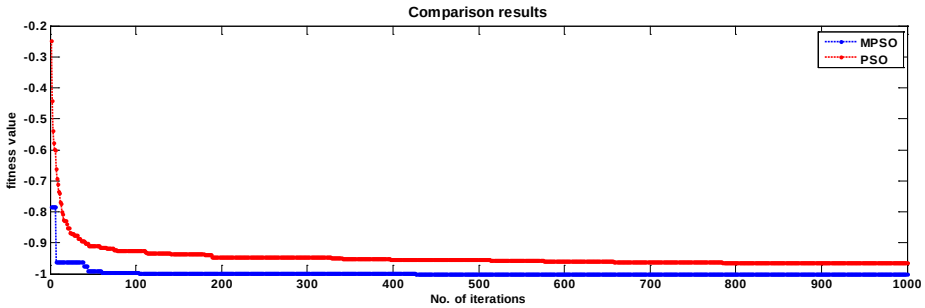

Fig. 5.2. Performance comparison of MPSO and PSO algorithms

In **Fig. 5.2**, the horizontal axis represents the number of iteration and the vertical axis represents the fitness values. The red line indicated the modified particles swarm optimization improvement. This means that when the two algorithms completed the same task, the modified PSO performs better than standard PSO. The plot results show that MPSO converges faster than the PSO algorithm. In each MPSO's iteration, the position and the velocity of all particles are updated and their fitness is evaluated together with their dimension number n. The fitness function complexity is based on the scheduling algorithm and also depended on the number of tasks i. The convergence time is influenced by the number of particles and the number of virtual resources considered. The numbers in **Table 5.2** indicate the average run time of best function values in 20 runs. Several experiments and different parameters were set to evaluate the efficiency of MPSO, so MPSO algorithm suits more to cloud computing.
Therefore, the average run time of the MPSO algorithm is shorter than that of PSO algorithm.

Table 5.2. Comparison results for 20 runs of particle swarm optimization algorithm technique

Algorithm	Fitness value	The number of iterations	Avg Run time(ms)
PSO	-0.2414	1000	3300
MPSO	-0.7934	1000	2044

Scenario 2: Analysis of the results based on CloudSim Simulations:
In this section, we considered to simulate our results with CloudSim platform [40] and to evaluate the performance of the proposed algorithm.
We considered some parameters such as completion time, execution time, and resource utilization. The process of classifying these parameters is known as Task parameterization. In our experiment, we used five virtual machines, and as well as ten virtual machines. The comparison experiments were carried out by using MPSO, PSO, and GA algorithms. We compared the performance of the three algorithm from task completion time, execution time, and resource utilization. We considered i number of tasks and j as the number of VMs. Tasks are the service requests in cloud computing environment and need to be allocated to VMs in order to be processed. The VMs have the configured processing capability such as processor, memory, and capacity size which are dynamically varied. We set task length between 1000MI to 15000MI with the proceeding speed of virtual machine 150MIPS to 300MIPS (MIPS is the million instructions per second of VMs) and (MI is the million instructions of tasks).
Other parameters used are from **Table5.1** above. The bellow figures show the variation of processing time of particles (tasks), execution time and resource utilization with respect to their number of VMs to complete the work. **Fig. 5.3** shows that the completion time decreases when the number of resources increases. The completion time of MPSO is shorter than PSO, and GA. This shows that MPSO has great advantage and can easily find very good solution space to reduce the processing time the tasks take to complete the process. **Fig. 5.3** shows the completion time for MPSO, PSO, and GA with respect to 10 virtual machines

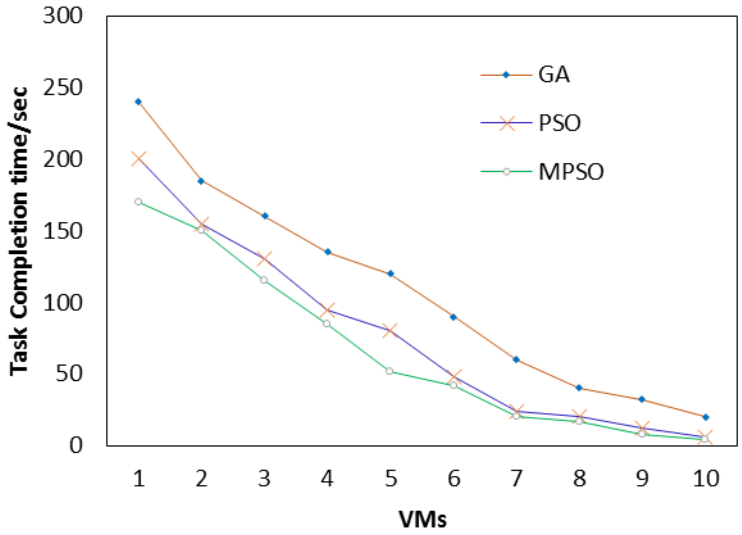

Fig. 5.3 Completion time for MPSO, PSO, and GA with respect to 10 virtual machines

To demonstrate the performance for the algorithms (GA, PSO, and MPSO), we use 5 virtual machines and 10 virtual machines for 10, 15, 25, 40 cloudlets.

Table 5.3. Task Scheduling comparison based on completion time (sec)

GA	PSO	MPSO	VM	Cloudlet
3.199166	2.162	1.7511	5	10
5.716983	3.799	2.8932		15
8.926751	7.367	5.6195		25
13.16712	8.496	5.3936		40

The results show that the MPSO algorithm outperforms the GA and PSO with the respect to the execution time.

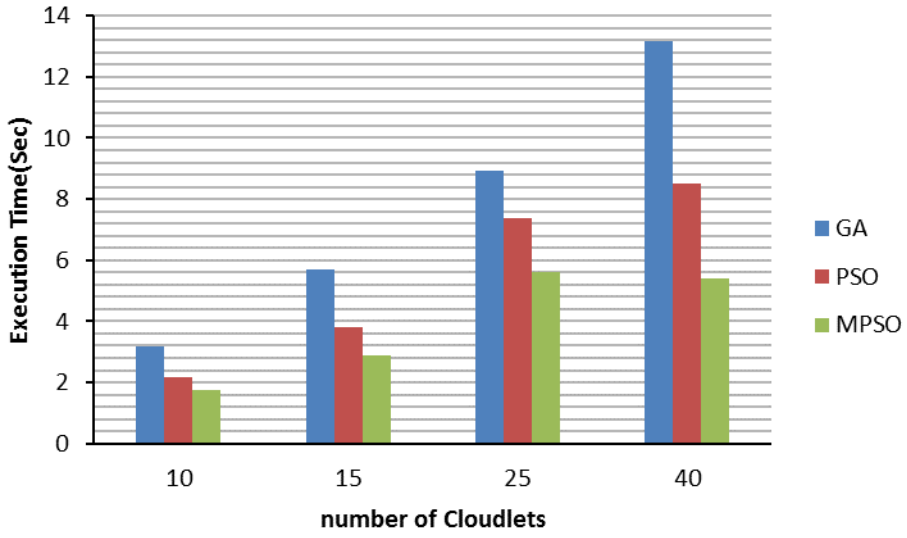

Fig.5.4 Representation of Execution Time for MPSO, PSO, and GA

The figure above shows the execution time for GA, MPSO, and PSO algorithms when using 5 virtual machines and setting 10, 15, 25, 40 cloudlets.

Table5.4. Task Scheduling comparison based on completion time(sec)

GA	PSO	MPSO	VM	Cloudlet
2.195	1.917	1.412	10	10
3.793	2.689	2.109		15
7.264	6.337	3.897		25
8.216	6.261	4.382		40

Based on the results in **Fig. 5.5** which shows the execution time of GA, MPSO, and PSO algorithms when using 10 virtual machines and various cloudlets, the proposed algorithm outperforms the current PSO and GA respectively.

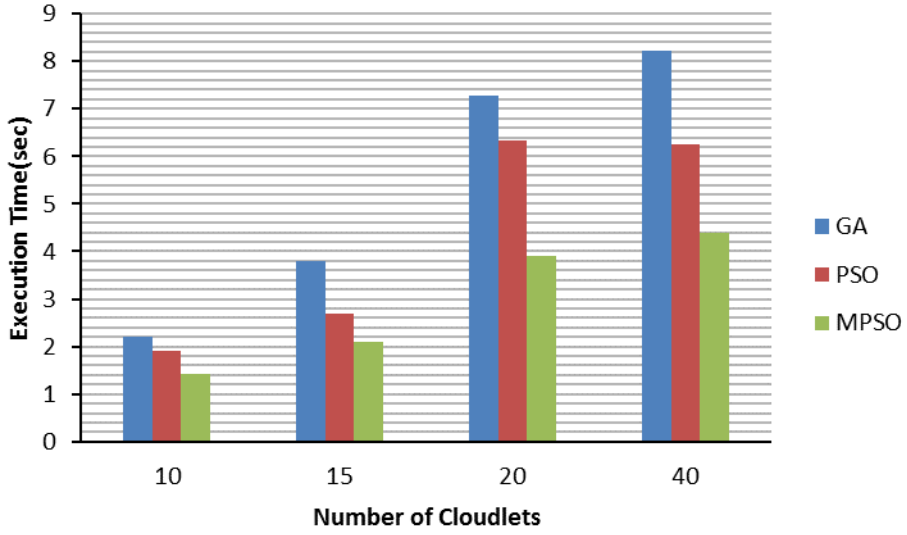

Fig. 5.5 Representation of Execution Time for MPSO, PSO, and GA when considering 10 VMs

GA	PSO	MPSO	VM	Cloudlet
0.615071	0.713026	0.850168	5	10
0.604812	0.728099	0.939132		15
0.607211	0.672281	0.940351		25
0.53191	0.58714	0.821439		40

Table 5.5 shows the resource utilization for GA, MPSO, and PSO algorithms while considering 5 virtual machines and 10, 15, 25, 40 cloudlets.
Based on the results in **Fig. 5.6**, MPSO algorithm outperforms PSO and GA with respect to the resource utilization for 5 VMs.

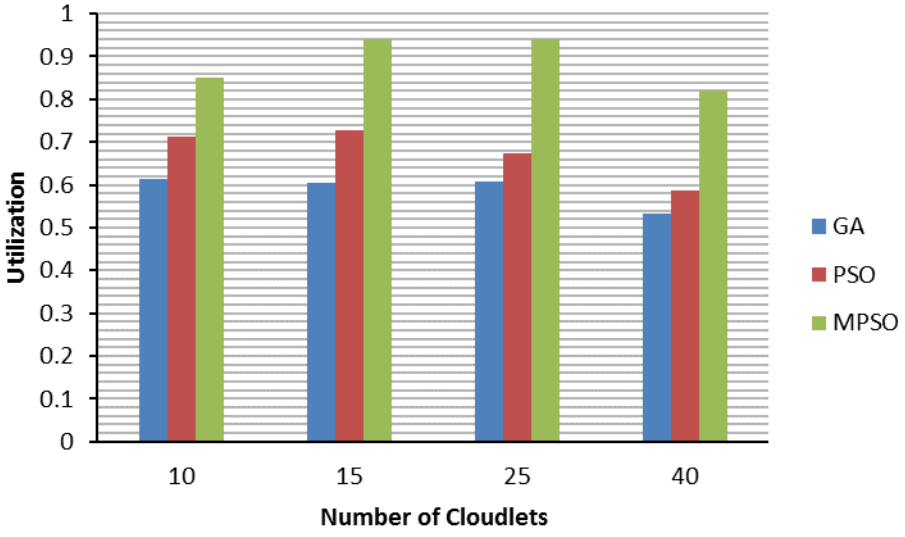

Fig. 5.6 The simulation results of the resource utilization for MPSO, PSO, and GA based on 5 virtual machines and various cloudlets.

Table 5.6. Comparison results of MPSO, PSO, and GA with respect to the resource utilization

GA	PSO	MPSO	VM	Cloudlet
0.4126	0.65016	0.7012	10	10
0.4399	0.69232	0.7893		15
0.3617	0.41951	0.7468		25
0.3861	0.70193	0.8806		40

Based on the results in **Fig. 5.7**, MPSO algorithm outperforms PSO, and GA with respect to the resource utilization while considering 10 virtual machines and 10, 15, 25, 40 cloudlets.

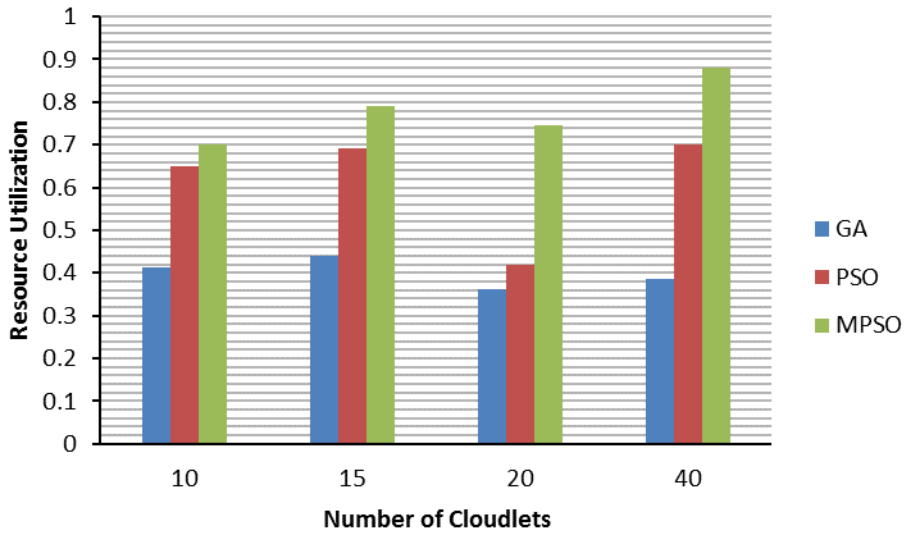

Fig. 5.7 Representation of resource utilization based on 10 VMs and various Cloudlets

The research shows how to use evolutionary techniques to enter a schedule as a search solution. We refer to the **Fig. 3.3** to show the mapping of tasks to the available resources. Considering the proposed algorithm, each particle represents a task for which it is randomly assigned to the available resource. The figures show the assignment of various cloudlets to five and ten virtual machines.

5. Conclusion

In a cloud environment, there are many tasks running on virtual resources, so this paper presents a modified particle swarm optimization algorithm applied to optimize task scheduling and virtual resource allocation in cloud computing with two constraints for time, cost, and resource utilization rate. From the simulations, it is shown that this algorithm could quickly and dynamically optimize virtual resources with a reduced total time for task scheduling in the cloud environment. The MPSO also can perform better than the PSO method proposed in the earlier researches in terms of processing time, cost and resource utilization rate. The PSO can fail to achieve the required optimum solution in cases when the problem to be solved seems to be complicated or complex but this can be fixed by using MPSO algorithm. Using MPSO algorithm in cloud computing resource allocation, the simulation results show that MPSO presents advantages in convergence speed, in finding global optimal, and in simplicity ability.

As we can see from the results, a MPSO optimization processes the best fitness value, decreases rapidly and converges while the number of iteration convergent to the higher value. To show how the evolution process is going on for both MPSO, the convergence of the average fitness values is also shown in **Fig. 5.2**, from which it is clear that MPSO seems to perform better. In the changing environment, like virtual cloud computing resources need to be operated in optimally manner. Therefore, the MPSO optimization algorithm can quickly allocate resources under the dynamic environment, and utilizes effectively the system resources to reduce cost, and makespan for the number of virtual resources and user jobs.

References

[1] Qi, H. and A. Gani, "Research on mobile cloud computing: Review, trend and perspectives," in *Proc. of Digital Information and Communication Technology and it's Applications (DICTAP), 2012 Second International Conference on*, IEEE, 2012. Article (CrossRef Link)

[2] Weng, C., et al., "The hybrid scheduling framework for virtual machine systems," in *Proc. of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, ACM, 2009.

[3] Mell, P. and T. Grance, The NIST definition of cloud computing, 2011. https://www.nist.gov/publications/nist-definition-cloud-computing

[4] Luo, Y. and B. Plale, "Hierarchical mapreduce programming model and scheduling algorithms," in *Proc. of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012)*, IEEE Computer Society, 2012. Article (CrossRef Link)

[5] Zhan, S. and H. Huo, "Improved PSO-based task scheduling algorithm in cloud computing," *Journal of Information & Computational Science*, 9(13): p. 3821-3829, 2012.

[6] Ning, W., et al., "A task scheduling algorithm based on qos and complexity-aware optimization in cloud computing," in *Proc. of Information and Communications Technology 2013, National Doctoral Academic Forum on*, IET, 2013. Article (CrossRef Link)

[7] Zhu, K., et al., "Hybrid genetic algorithm for cloud computing applications," in *Proc. of Services Computing Conference (APSCC), 2011 IEEE Asia-Pacific*, IEEE, 2011. Article (CrossRef Link)

[8] Lin, S.-W., K.-C. Ying, and C.-Y. Huang, "Multiprocessor task scheduling in multistage hybrid flowshops: A hybrid artificial bee colony algorithm with bi-directional planning," *Computers & Operations Research*, 40(5), p. 1186-1195, 2013. Article (CrossRef Link)

[9] Al-maamari, A. and F.A. Omara, "Task Scheduling Using PSO Algorithm in Cloud Computing Environments," *International Journal of Grid and Distributed Computing*, 8(5): p. 245-256, 2015. Article (CrossRef Link)

[10] Xu, S.-H., et al., "A Combination of Genetic Algorithm and Particle Swarm Optimization for Vehicle Routing Problem with Time Windows," *Sensors*, 15(9), p. 21033-21053, 2015. Article (CrossRef Link)

[11] Karaboga, D., Artificial bee colony algorithm. scholarpedia, 5(3), p. 6915, 2010.

[12] Kumar, R.S. and S. Gunasekaran, "Improving task scheduling in large scale cloud computing environment using artificial bee colony algorithm," *International Journal of Computer Applications*, 103(5), 2014. Article (CrossRef Link)

[13] Akkoyunlu, M.C., O. Eng n, and K. Büyuközkan, "A harmony search algorithm for hybrid flow shop scheduling with multiprocessor task problems," in *Proc. of Modeling, Simulation, and Applied Optimization (ICMSAO), 2015 6th International Conference on*, IEEE, 2015. Article (CrossRef Link)

[14] Mirjalili, S. and S.Z.M. Hashim, "A new hybrid PSOGSA algorithm for function optimization," in *Proc. of Computer and information application (ICCIA), 2010 international conference on*, IEEE, 2010. Article (CrossRef Link)

[15] Lili, X., et al., "An improved binary PSO-based task scheduling algorithm in green cloud computing," in *Proc. of 2014 9th International Conference on Communications and Networking in China*, 126-31, 2014. Article (CrossRef Link)

[16] Agrawal, S. and R. Shimpi, Modified Particle Swarm Optimization.
[17] Labed, S., A. Gherboudj, and S. Chikhi, "A modified hybrid particle swarm optimization algorithm for multidimensional knapsack problem," *Int. J. Comput. Appl*, 34(2): p. 1, 2011. Article (CrossRef Link)
[18] Khan, S.U., et al., "A Modified Particle Swarm Optimization Algorithm for Global Optimizations of Inverse problems," *IEEE Transactions on Magnetics*, Vol. 52, Issue 3, 2016. Article (CrossRef Link)
[19] Pandey, S., et al., "A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments," in *Proc. of Advanced information networking and applications (AINA), 2010 24th IEEE international conference on*, IEEE, 2010. Article (CrossRef Link)
[20] Peyvandi, M., M. Zafarani, and E. Nasr, "Comparison of Particle Swarm Optimization and the genetic algorithm in the improvement of power system stability by an SSSC-based controller," *Journal of Electrical Engineering and Technology*, 6(2): p. 182-191, 2011. Article (CrossRef Link)
[21] Sridhar, M., "Hybrid Genetic Swarm Scheduling for Cloud Computing," *Global Journal of Computer Science and Technology*, 15(3), 2015.
[22] Shi, P., et al., "Dependable Deployment Method for Multiple Applications in Cloud Services Delivery Network," *[J]. China Communications*, 8(4): p. 65-75, 2011.
[23] Guo, L., et al., "Task scheduling optimization in cloud computing based on heuristic algorithm," *Journal of Networks*, 7(3): p. 547-553, 2012. Article (CrossRef Link)
[24] Jena, R., "Multi objective task scheduling in cloud environment using nested PSO framework," *Procedia Computer Science*, 57: p. 1219-1227, 2015. Article (CrossRef Link)
[25] Hsu, Y.-C., P. Liu, and J.-J. Wu, "Job sequence scheduling for cloud computing," in *Proc. of Cloud and Service Computing (CSC), 2011 International Conference on*, IEEE, 2011. Article (CrossRef Link)
[26] Priyadarsini, R.J. and L. Arockiam, "PBCOPSO: A parallel optimization algorithm for task scheduling in cloud environment," *Indian Journal of Science and Technology*, 8(16), 2015. Article (CrossRef Link)
[27] Eberhart, R.C. and Y. Shi, "Comparison between genetic algorithms and particle swarm optimization," in *Proc. of International Conference on Evolutionary Programming*, Springer, 1998. Article (CrossRef Link)
[28] Hassan, R., et al., "A comparison of particle swarm optimization and the genetic algorithm," in *Proc. of the 1st AIAA multidisciplinary design optimization specialist conference*, 2005. Article (CrossRef Link)
[29] Jones, K.O., "Comparison of genetic algorithm and particle swarm optimization," in *Proc. of Int. Conf. Computer Systems and Technologies*, 2005.
[30] Elbeltagi, E., T. Hegazy, and D. Grierson, "Comparison among five evolutionary-based optimization algorithms. Advanced engineering informatics," 19(1): p. 43-53, 2005. Article (CrossRef Link)
[31] De Jong, K.A. and W.M. Spears, "Using Genetic Algorithms to Solve NP-Complete Problems," *in ICGA*, 1989.
[32] Xu, L., et al., "An improved binary PSO-based task scheduling algorithm in green cloud computing," in *Proc. of Communications and Networking in China (CHINACOM), 2014 9th International Conference on*, IEEE, 2014. Article (CrossRef Link)
[33] Pooranian, Z., et al., "An efficient meta-heuristic algorithm for grid computing," *Journal of Combinatorial Optimization*, 30(3): p. 413-434, 2015. Article (CrossRef Link)
[34] Kennedy, J., "Particle swarm optimization," *Encyclopedia of Machine Learning*, Springer, p. 760-766, 2010. Article (CrossRef Link)
[35] Eberhart, R. and J. Kennedy, "A new optimizer using particle swarm theory," in *Proc. of Micro Machine and Human Science, 1995. MHS '95., Proceedings of the Sixth International Symposium on*, 1995. Article (CrossRef Link)

[36] Chapman, B., "When clouds become green: the green open cloud architecture," *Parallel Computing: From Multicores and GPU's to Petascale*, 19, p. 228, 2010. Article (CrossRef Link)
[37] Sedighizadeh, M., et al., "Parameter optimization for a PEMFC model with particle swarm optimization," *Int J Eng Appl Sci*, 3, p. 102-108, 2011.
[38] Liu, C.-Y., C.-M. Zou, and P. Wu, "A task scheduling algorithm based on genetic algorithm and ant colony optimization in cloud computing," in *Proc. of Distributed Computing and Applications to Business, Engineering and Science (DCABES), 2014 13th International Symposium on*, IEEE, 2014. Article (CrossRef Link)
[39] Rostami, A. and M. Lashkari, "Extended PSO algorithm for improvement problems K-Means clustering algorithm," *International Journal of Managing Information Technology*, 6(3), p. 17, 2014. Article (CrossRef Link)
[40] Calheiros, R.N., et al., "CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Software - Practice and Experience*, 41(1), p. 23-50, 2011. Article (CrossRef Link)

Frederic Nzanywayingoma was born in Rwanda, September, 1981. He received his B.S. degree in Electronic and Communication Systems Engineering from National University of Rwanda in 2010 and his M.S. degree in Information Communication Engineering from University of Science and Technology Beijing, China in 2013. Currently, he is a PhD candidate at the same university. His research interests include Machine to Machine Communications, Cloud Computing, Scheduling Algorithms and Optimization Methods.

Professor Yang Yang graduated from Beijing Iron and Steel Institute of Automation System in 1982. He received his PhD in Information Engineering, from University of Science and Technology in Lillie, France, in 1988. He has been a professor at University of Science and Technology Beijing since 1988. He has published in journals and conferences both at home and abroad more than 200 papers, completed books. His research interests include Service Science and Cloud Computing, Intelligent Control, Image Processing and Pattern Recognition, Multimedia Communication, Grid Technology.