Hybrid Task Scheduling Method for Cloud Computing by Genetic and PSO Algorithms

Amin Kamalinia
Department of Computer Engineering, Urmia Branch, Islamic Azad University, Urmia, Iran
Amin.kamalinia@gmail.com
Ali Ghaffari*
Department of Computer Engineering, Tabriz Branch, Islamic Azad University, Tabriz, Iran
a.ghaffari@iaut.ac.ir

Received: 11/Sep/2015 Revised: 17/Aug/2016 Accepted: 28/Sep/2016

Abstract

Cloud computing makes it possible for users to use different applications through the internet without having to install them. Cloud computing is considered to be a novel technology which is aimed at handling and providing online services. For enhancing efficiency in cloud computing, appropriate task scheduling techniques are needed. Due to the limitations and heterogeneity of resources, the issue of scheduling is highly complicated. Hence, it is believed that an appropriate scheduling method can have a significant impact on reducing makespans and enhancing resource efficiency. Inasmuch as task scheduling in cloud computing is regarded as an NP complete problem; traditional heuristic algorithms used in task scheduling do not have the required efficiency in this context. With regard to the shortcomings of the traditional heuristic algorithms used in job scheduling, recently, the majority of researchers have focused on hybrid meta-heuristic methods for task scheduling. With regard to this cutting edge research domain, we used HEFT (Heterogeneous Earliest Finish Time) algorithm to propose a hybrid meta-heuristic method in this paper where genetic algorithm (GA) and particle swarm optimization (PSO) algorithms were combined with each other. The experimental results of simulation are shown that the proposed algorithm optimizes the average makespans of the HEFT_UpRank, HEFT_DownRank, HEFT_LevelRank and MPQMA for 100 independent task graphs scheduling with 10, 50 and 100 tasks. Total optimization of makespans by the proposed algorithm against the other algorithms were 6.44, 10.41, 6.33 and 4.8 percent respectively.

Keywords: Cloud Computing; Task Scheduling; Genetic Algorithm; Particle Swarm Optimization Algorithm.

1. Introduction

In the recent years, with huge advancement in IT (information technology) based systems [1-3], cloud computing is considered as one of the most important trends [4]. Cloud computing is considered to be a novel scientific tool and asset for high-performance computing (HPC). It refers to a technology which uses internet and central distant service provision in order to maintain data and applications. Moreover, this technology can be used in high-performance computing to centralized storages, memory, processing and bandwidth. Cloud computing is used as a technology to supply the resources of information and communication technology (ICT) dynamically and scalably all over the internet. Also, cloud computing is a pattern which provides computational resources are delivered to users on demand over the Internet as a public service [5]. Furthermore, Task scheduling in software defined networks (SDNs) [6] based cloud computing is an important challenges for future work. Scheduling is regarded as a decision-making process which is regularly used in the majority of production and service-providing industries which is used to enhance efficiency optimization [7]. Indeed, scheduling refers to the allocation of limited resources to tasks throughout time [8]. It should be noted that unique features and characteristics of resource management and service scheduling distinguish cloud computing from other computing methods. Whereas centralized scheduling in a clustered system is aimed at enhancing the efficiency of the entire system, distributed scheduling in a grid computing system is intended to enhance efficiency for a certain final user. When compared with other systems, scheduling in cloud computing is much more complicated; hence, a centralized scheduling is required [9]. Each cloud provider is obliged to provide services for the users. It should be noted that the cloud provider provides services without mentioning the location of host infrastructures and data centers. On the other hand, commercial features make it necessary for cloud computing to consider the users' needs and preferences with respect to the quality of services all over the world.

In cloud computing, there is a data center which includes interconnected equipment and machines where they have high-speed links and connections with each other. Such an environment is appropriate for processing a mass of diverse tasks and activities. Scheduling in distributed systems refers to the allocation of multiple tasks to multiple machines which intends to enhance optimization; hence, it is considered to be an NP-complete. It can be argued that heuristic algorithms are usually used as less than ideal and desirable algorithms to

* Corresponding Author

achieve relatively good solutions. Hence, in recent years, evolutionary algorithms are used to better optimize solutions.In this paper, a novel algorithm has been proposed where genetic and particle swarm optimization algorithms were combined and also the HEFT algorithm was used to schedule tasks in the context of cloud computing. Also, we simulated and evaluated proposed scheme and analyzed it statistically. The results of simulation and statistical analysis of proposed method indicate that the proposed algorithm is optimized the average makespans of the HEFT_UpRank, HEFT_DownRank, HEFT_LevelRank and MPQMA for 100 independent task graphs scheduling with 10, 50 and 100 tasks. Percent of the total optimization of makespans for the mentioned algorithms were 6.44, 10.41, 6.33 and 4.8 respectively.

The reminder of the paper is organized as follows: in Section 2, studies related to task scheduling are briefly reviewed. In Section 3, HEFT algorithm is described and discussed. In Section 4, genetic algorithm (GA) is described and reviewed. Section 5 is concerned with PSO algorithm. Section 6 describes the algorithm proposed in this paper. Section 7 describes experimental results of the proposed algorithm. Finally, Section 8 presents the conclusion and suggestions for further research.

2. Related Works

Most of studies on task scheduling issues has been studied in distributed high performance computing (HPC) environments such as clusters, Grid [10] and also cloud computing. Numerous research studies have been conducted on the issue of scheduling in cloud computing. Some of the related studies are reviewed in this section of the study. Researchers considered the virtualization features and commercial features in cloud computing to propose a task scheduling algorithm for the first time based on Berger model [11]. This algorithm maintains the dual fairness limitation in the process of task scheduling. The first categorization limitation selects the user's tasks based on service quality priorities by creating a public wait function. In selecting a user's tasks, task categories are taken into consideration to avoid the fairness of resources in the selection process. The second limitation is related to define resource justice function which is used to judge about the justice and fairness of resource allocation. The main motivation of researchers in [12] was to design and develop a cloud resource server for efficient handling of cloud resources and doing tasks for scientific programs with respect to the deadline determined by the user. The deadline was based on task scheduling. Task scheduling was combined and implemented with particle swarm optimization algorithm. This solution was intended to reduce task execution time and cost based on the defined fitness function. Researchers developed a new task scheduling algorithm for executing massive programs and applications in cloud [13]. This economical and low-cost task scheduling algorithm operates based on two heuristic methods. The first strategy dynamically maps the tasks to the best virtual machines in terms of cost according to the Pareto dominance. The second strategy which complements the first strategy reduces financial costs from unimportant tasks.

Researchers in [14], proposed a novel memetic task scheduling algorithm on cloud environment using multiple priority queues which named MPQMA (multiple priority queues and a memetic algorithm). This algorithm employs a genetic algorithm with local search algorithm to solve scheduling problem in heterogeneous computing systems. The main goal of this algorithm is using advantage of MA to increase the convergence speed of the solutions. Experimental results of randomly generated graphs discovered that the MPQMA algorithm optimized the other four current algorithms in terms of makespan with fast convergence of solutions. In work [15], the researchers proposed a population based meta-heuristic algorithm based on particle swarm optimization (PSO) to schedule applications on cloud resources. This algorithm considers both computation cost and data transmission cost. Experiment results are gained with a workflow application by varying its computation and communication costs. The algorithm is compared with existing 'Best Resource Selection' (BRS) algorithm in terms of the cost savings. The results illustrated that PSO betters BRS in times cost savings and distribution of workload onto resources. In research [16], the concept of project scheduling with the workflow scheduling problem are integrated to formulate a mathematical model that aims to minimize the makespan. In order to solve the workflow scheduling optimization problem two Artificial Bee Colony algorithms are applied. This algorithm is compared with the optimal solutions obtained by Gurobi optimizer to evaluate performance of ABC on the different workflows. The experimental results depict that ABC can be utilized as a practical method for complex workflow scheduling problems in the cloud computing environment.

In [17], a task scheduling based on Ant Colony Optimization (ACO) for task scheduling problem is proposed to minimize the makespan of the tasks submitted on the cloud environment. In addition, the ACO is applied to improve the efficiency of Cloud computing system. Experimental results are achieved by Cloud simulator which called CloudSim. In this work, the various graph from 100 to 500 of tasks are used to evaluate the algorithm in different situations.

Using genetic algorithm and multiple priority queues called MPQGA, the researchers proposed a task scheduling method in heterogeneous computational systems [18]. The rationale behind this method was to benefit from both heuristic and evolutionary algorithms and make up their shortcomings. The algorithm proposed in [18] utilized the genetic algorithm for allocating task priority and made use of the EFT heuristic method for mapping and dedicating tasks to the processor. In MPQGA

method, crossover and mutation operators, and the appropriate fitness function were designed for the scenario of directed acyclic graph. The results of experiments indicated that the MPQGA algorithm performed better than the two non-evolutionary methods and the random search method with respect to scheduling quality.

3. HEFT Algorithm

In general task scheduling algorithms divided into static or dynamic [19]. The static task scheduling algorithm HEFT was first introduced in [20]. In this method, the scheduling algorithm was used for a limited number of heterogeneous processors. It was used for parallelizing the processors so as to enhance efficiency and fasten scheduling. Before discussing the HEFT algorithm, it is necessary to introduce the terms EFT (earliest finish time) and EST (earliest start time). EST and EFT refer to the earliest starting time and the earliest finishing time of the execution of the task n_i on the processor p_j. The value of EST for then try task is equal to zero which has been defined in Equation (1). For other tasks in the graph, the values of EST and EFT are defined recursively according to Equations (2) and (3). For measuring the EFT of task n_i, all the procedures of this task should be scheduled. In these equations, *pred*(n_i) stands for the entire procedures of the task n_i and $avail\{j\}$refers to the earliest time of the p_j processor which is ready to execute the task. If n_k is a recent task which is dedicated to the processor p_j, then, $avail\{j\}$refers to the time at which the processor pj has finished the execution of task nk and it is ready to execute another task in case it has used a non-insertion-based scheduling method. Internal *max* in Equation (2) measures the time at which all the required data for n_i has arrived at p_j. After task n_m has been scheduled on the processor p_j, the earliest starting time and the earliest finishing time of task n_m on the processor p_j will be equal to AST (actual start time) and AFT (actual finishing time). It should be noted that, according to Equation (5), AFT will be equal to the smallest obtained EFT for that task. After the scheduling of all the graph tasks, the scheduling makespan will be equal to the AFT of the exit task. In case there are several output tasks or in case there are no pseudo-task, the makespan of the scheduling will be obtained through Equation (4). Moreover, $c_{m,i}$ refers to the communication costs from node m to node *i*. If the two tasks m and *i* are allocated to the same processor, $c_{m,i}$ will be equal to zero.

$$EST(n_{entery}, p_j) = 0 \quad (1)$$

$$EST(n_i, p_j) = max\left\{avail\{j\}, \max_{n_m \in pred(n_i)} \left(AFT(n_m) + c_{m,i}\right)\right\} \quad (2)$$

$$EFT(n_i, p_j) = \omega_{i,j} + EST(n_i, p_j) \quad (3)$$

$$makespan = max\{AFT(n_{exit})\} \quad (4)$$

$$AFT(n_i, p_j) = \min_{1 \le l \le m} EFT(n_i, p_l) \quad (5)$$

In the HEFT algorithm, the priorities are determined recursively based on task upward rank according to Equation (7). In this equation, *succ*(n_i) refers to a set of the successors of task n_i and $\overline{c_{i,j}}$ stands for the average cost of the communication edge (i,j) and $\overline{\omega_i}$ refers to the average computational cost of task ni which is measured through Equation (8). As its name denotes, since rank starts from the output node and is measured recursively, hence, it is referred to as upward rank. The upward rank of the output node is measured through Equation (6). Basically, $rank_u(n_i)$ refers to the length of critical path from task n_i to the output task which also includes the computational cost of task n_i.

$$rank_u(n_{exit}) = \overline{\omega_{exit}} \quad (6)$$

$$rank_u(n_i) = \overline{\omega_i} + \max_{n_j \in succ(n_i)} \left(\overline{c_{i,j}} + rank_u(n_j)\right) \quad (7)$$

$$\overline{\omega_i} = \sum_{j=1}^{q} \omega_{i,j}/q \quad (8)$$

Similarly, downward rank is obtained recursively through Equation (9). $pred(n_i)$ refers to the set of procedures of the task n_i. As the name suggests, downward rank is obtained recursively through the downward graph movement of the task which starts from the input node of graph. The downward rank of the input node is equal to zero. In general, $rank_d(n_i)$ is the longest distance from the input node to the task n_i where the computational cost of the task is not considered.

$$rank_d(n_i) = \max_{n_j \in pred(n_i)} \left(rank_d(n_j) + \overline{\omega_j} + \overline{c_{i,j}}\right) \quad (9)$$

The HEFT algorithm has two phases. The first phase is concerned with prioritization of tasks so that the priorities of all tasks are measured. The second phase is concerned with the selection of the processor so that task are chosen based on their priorities and the scheduling of each selected task is allocated to the best processor which can minimize the finishing time of the task.

Task prioritization phase: in this phase, the priority of each task can be measured through different methods some of which are mentioned below. The priority of each task is determined through upward rank and downward rank according to the procedure reported in [20] which have been defined in Equations (7) and (9). Also, priorities can be measured by combining the two methods which have been described in [18] in which Equation (10) is first used to level the graph; then, the values of levels and the values of upward rank and downward rank are used to produce a new prioritization queue. As a result, those tasks which are at the same level are arranged in a descending order. After one of the mentioned methods is selected and their values for each task are calculated, a list of tasks is produced based on descending order of tasks. In case the value of the selected method is equal for several tasks, the tasks are randomly

selected. It should be noted that upward rank is based on average computation and communication cost. It is obvious that the descending order of the upward rank values create a topological order of tasks which is regarded as a linear order in which precedence limitations are preserved.

$$Level(T_i) = \begin{cases} 0, & if T_i = T_{entry}; \\ \max\limits_{T_j \in pred(T_i)} \left(Level(T_j) \right) + 1, & otherwise \end{cases} \quad (10)$$

Processor selection phase: in the majority of task scheduling algorithms, the earliest time for the accessibility of the processor p_j for executing a task is when p_j has finished the previous task. Moreover, some algorithms have the insertion-based policy. As a case in point, HEFT is based on insertion-based policy which considers the probability of inserting a task in the idle time slot between two previous scheduled tasks. The length of the idle time slot of the processor is the distance between the starting execution time and the finishing time of two tasks which were consecutively executed on the same processor. At least, it should be able to execute the computational cost of the task. Furthermore, scheduling an idle time slot should consider the precedence limitations.

4. Genetic Algorithm

Genetic algorithm is deemed to be a search and optimization method which is based on the principles of genetics and natural selection [21]. Genetic algorithm is a type of evolutionary algorithms which has been inspired by the Darwin theory about evolution. This algorithm was developed by John Holland at the Michigan University during the 1960's and 1970's. Later, one of Holland's students named David Goldberg was able to propose a solution based on evolutionary algorithms to a challenging issue about the control of gas pipeline transmission [22,23]. The major contribution of Holland was published in a book entitled "Adaptation in Natural and Artificial Systems" [24]. Holland's theory was expanded and now it was developed into a powerful algorithm for solving the search and optimization problems. This algorithm has the following three operators: selection, crossover and mutation operators. The details about the implementation of these operators have been discussed later in this paper.

5. PSO Algorithm

Particle swarm optimization (PSO) algorithm is a population-based random optimization method which was proposed by Russell Eberhart and James Kennedy in 1999. The development of this algorithm was inspired from the swarm behavior of birds or fish [24,25]. This system begins with a population which has random solutions and it updates the generation to find an optimal solution. In contrast with genetic algorithm, none of the evolutionary operators such as crossover and mutation are available in the PSO algorithm. Solutions in PSO algorithm are referred to as particles which move in the problem search space and follow the current optimal particle [26]. In this algorithm, each particle follows the particle which has a better fitness function among all the particles. However, it does not forget its own experience. Hence, it follows the condition and state in which it has the best fitness function. Thus, in each iteration of the algorithm, each particle determines its next position based on two values: first, the best position that the particle has ever had indicated by pbest and also the best position that all the particles have ever had indicated by gbest. In other words, gbest refers to the best pbest in the entire population. Conceptually, pbest for each particle refers to the memory which a particle has experienced about its best position. gbest represents the public knowledge of the population and when particles change their positions based on gbest, they try to keep up with the knowledge of the population. Conceptually, the best particle connects all the particles of the population with each other [26,27]. In this method, the next position for each particle is determined according to the following equation:

$$v_i(t+1) = w.v_i(t) + c_1 r_1 \left(pbest(t) - x_i(t) \right) + c_2 r_2 \left(gbest(t) - x_i(t) \right) \quad (11)$$

$$x_i(t+1) = x_i(t) + v_i(t+1) \quad (12)$$

In Equation (11), $v_i(t)$ refers to the speed or velocity of particle i in the time unit of t. Also, w which is indicated byα refers to the coefficient or inertia weight for controlling exploitation and exploring the search space. C_1 and C_2 are the learning parameters. In other words, they are constant accelerators which change the speed changes of the particle towards pbest and gbest. Indeed, the value of these two variables are equal to 2.The values of r_1 and r_2 are two random variables which vary between 0 and 1. In Equation (12), $x_i(t)$ represents the position of particle i in the time unit of t.

6. The Proposed Algorithm

The input of the problem is a directed acyclic graph which is indicated by $G = (V, E)$. Each node is a member of V set which is a vertex of the graph and indicates one task from all the set of tasks; the weight of these nodes determine the execution time of the tasks. This graph also includes a set of edges; in other words, it includes E which indicates the prerequisite relations among the tasks. In case there exists an edge such as (t_i, t_j), it means that task t_j cannot start until task t_i is finished. These edges are weighted and the weight of each edge indicates the communication cost of sending a message between two tasks. This cost exists when two related tasks are executed on different processors or machines and in case they are executed on the same processor or machine, the cost of communication between them will be zero.

Directed acyclic graph illustrated in Figure 1 includes the following tasks: $t_0, t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8, t_9, t_{10}$ which are the input of the proposed algorithm. The node t_0 is the entry task and t_{10} is the exit task. Table 1 indicates the costs of executing tasks on the m_0, m_1 and m_2. Also, $\overline{\omega}$ indicates the average costs of executing tasks on the machines. As noted, each task is executed with a different cost on each machine which indicates the heterogeneity of the computational context of tasks.

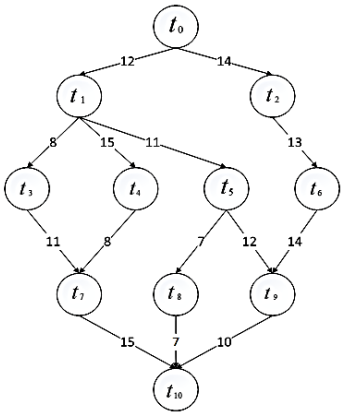

Fig. 1. DAG with 11 tasks

Table 1. Task execution costs on machines

Tasks	m_0	m_1	m_2	$\overline{\omega}$
t_0	7	9	8	8
t_1	10	9	14	11
t_2	5	7	6	6
t_3	6	8	7	7
t_4	10	8	6	8
t_5	11	13	15	13
t_6	12	15	18	15
t_7	10	13	7	10
t_8	8	9	10	9
t_9	15	11	13	13
t_{10}	8	9	10	9

One of the most important challenges in scheduling tasks in the cloud computing context is the selection of the best solutions for allocating resources to the tasks so that the cost and task finishing time are reduced. Inasmuch as there are a lot of tasks and there are different solutions for different tasks, hence, the selection of a solution is not a unique choice. That is, there is a set of choices and each choice is not preferred to the other choice. In the proposed hybrid method in this paper, a set of answers is produced by the genetic algorithm; then, these answers are considered as the initial population for the PSO algorithm and based on these answers, the next population for the genetic algorithm is produced with the help of PSO algorithm. At the end of this stage, based on the PSO algorithm, the whole produced answers are updated and the stages are repeated again. In each repetition, first, the particles find answers with respect to the operators of mutation and crossover. Then, PSO algorithm is used to produce children without moving entry and exit nodes. Hence, an optimal population is produced. It should be noted that if the children's priority is violated after the production of children, they will be sorted from left to right so that the priorities are not violated. In the proposed algorithm, the solutions prevented premature convergence before achieving an absolute optimal solution. It should be noted that after the crossover and mutation operators are executed each time, the replacement process is carried out so that the produced children are compared with their parents. If the fitness function of the children are not better than their parents, then, they are eliminated. Otherwise, they will replace and eliminate their parents. Figure 2 illustrates the flowchart of the genetic and PSO algorithms proposed in this paper.

In the PSO algorithm, each particle includes a solution which cover the context of the problem. In each iteration, the fitness or cost function is measured for all the particles. Then, the memory of each particle (pbest) is compared with the obtained value and in case the value of particle cost function is smaller than the value of its memory, particle memory will be equal with the current state of that particle; if these conditions occur, then, in this way, the memory value will be compared with the gbest value. As a result, the minimum solution is obtained for the problem. The implementation of the PSO algorithm is considered to be computationally simple; in case appropriate values are used for its parameters, it is highly probable to find an optimal answer. To avoid local optimality, the PSO algorithm functions in a way that when it is placed in an optimality, the particles mutate to other parts of the search space. Then, in other parts, they search for optimal answers.

In the genetic algorithm, once the initial population is created, the appropriateness of the answers is measured by means of the fitness function value. For having an optimal answer in the proposed method, the proposed model should have a small value for the fitness function.

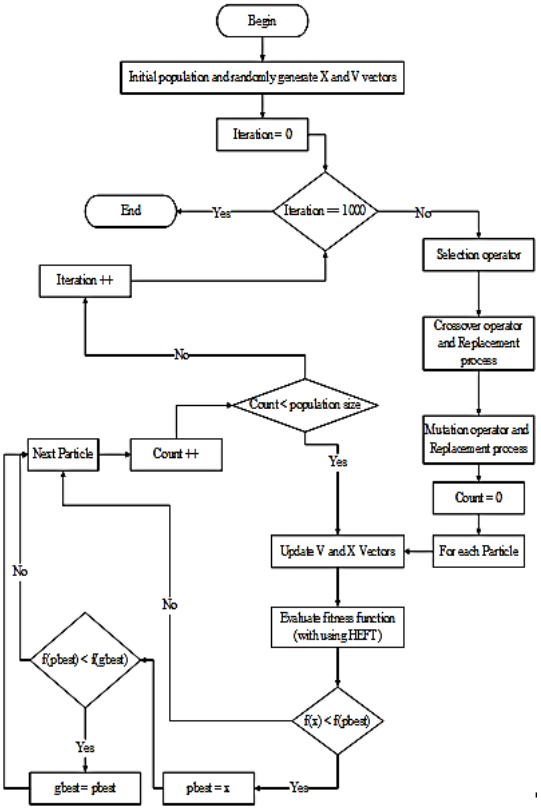

Fig. 2. Flowchart of the proposed algorithm in this paper

6.1 The Production of Initial Population

The initial population includes particles which are independent of each other where the sizes of chromosomes are fixed. In this paper, for having variety and appropriate initial values, three traditional heuristic methods were used to give initial values to the three particles. The three methods include upward rank, downward rank and a combination of these two methods based on their rank [18]. The initial values of the three particles were given according to the above-mentioned methods for the graph included in Figure 1. Indeed, multiple priority queues are produced which is shown in Table 2 for the directed acyclic graph. The remaining particles were randomly valued which is explained later in the paper. That is, the beginning and end of the chromosome which are the start and exit nodes are established in the chromosome. Those between these two nodes are randomly selected from left to right and are sorted provided that the priorities are not violated.

Table 2. Task Priorities

Tasks	$rank_u(t_i)$	$rank_d(t_i)$	*level*	$rank_u(t_i) + rank_d(t_i)$
t_0	102	0	0	102
t_1	79	20	1	99
t_2	80	22	1	102
t_3	52	39	2	91
t_4	50	46	2	96
t_5	57	42	2	99
t_6	61	41	2	102
t_7	34	62	3	96
t_8	25	62	3	87
t_9	32	70	3	102
t_{10}	9	93	4	102

6.2 Measuring Makespan for each Particle

For measuring makespan for each particle in this paper, tasks should be executed based on a processor or machine allocation method. This operation was conducted by means of the HEFT processor allocation method on each particle which is discussed below.

6.3 Fitness Function

The fitness value plays a significant role in deciding which particles should be used to produce the next generation. In this paper, makespan of a DAG is obtained from finishing time of exit task in an application which this makespan assumed the fitness of algorithm. In scheduling issue, the purpose of allocating task is to reduce the makespan without violating priorities. The makespan is obtained through Equation (4) and the fitness function is obtained through Equation (13).

$$fitness_i = makespan_i \tag{13}$$

6.4 Selection Operator

One of the significant parts of genetic algorithm is selection which has a remarkable impact on convergence. Indeed, a particle with a better fitness value is more likely to mate. One of the best implementation methods is the roulette wheel. This method assumes that the selection probability is a ratio of particle fitness. Some of the particles will be reselected for the genetic operation based on their fitness. A particle with the highest fitness is highly probable to be selected. Particles are measured according to their fitness. The value of the fitness function is always greater than zero. p_i stands for the probability of each particle to be selected is measured through Equation (14). Algorithm 1 shows the selection pseudo code.

$$p_i = \frac{fitness_i}{\sum_{j=1}^{PopSize} fitness_j} \tag{14}$$

Algorithm 1. Roulette wheel pseudo code

1: Generate a random number $R \in [0,1]$
2: **For** i=1 to PopSize **do**
3: **If** $p_i > R$ **then**
4: Select the chromosome;
5: Return the chromosome;
6: **end if**
7: **end for**.

Algorithm 2. Pseudo code of single-point combination operator

1: Choose randomly a suitable crossover point i;
2: Cut the first parent's chromosome and the second parent's chromosome into left and right segments
3: Generate a new offspring, namely the child one;
4: Inherit the left segment of the first parent's chromosome to the left segment of the child one's chromosome;
5: Copy genes in second parent's chromosome that do not appear in the left segment of first parent's chromosome to the right segment of child one's chromosome;
6: Generate a new offspring, namely the child two;
7: Inherit the left segment of the second parent's chromosome to the left segment of the child two's chromosome;
8: Copy genes in first parent's chromosome that do not appear in the left segment of second parent's chromosome to the right segment of child two's chromosome;
9: **if** offspring's fitness values are better than their parents **then** replace them
10: **if** step 9 is true **then** compare fitness value of offspring with local best **if** offspring's fitness value is better **then** replace it.
11: **if** step 10 is true **then** compare fitness value of offspring with global best **if** offspring's fitness value is better **then** replace it.

6.5 Crossover Operator

The population of a genetic algorithm is evolved and completed by crossover and mutation. In the method used in this paper, the crossover operator is regarded as a significant operation. Crossover is a function of replacing some genes of one parent with genes of another parent. In the task scheduling issue, the crossover operator combines the two parents with each other so as to produce two valid children.

In this paper, single-point crossover was implemented according to the method mentioned in [18]. That is, firstly, a random point between l and n is selected and the crossover point takes the priority queue of both parents from left to right in case they are not identical. For example, consider the particles depicted in Figure 3. The crossover point which is equal to 6 produces the single point of two new children. Indeed, it uses crossover

operator to replace some genes. The left part of children inherit their parents' genes. Then, some selected genes are eliminated from the parent and the remaining genes are added to the child from left to right. Consequently, the child will also be valid [18]. Then, the value of fitness function will be measured for each child. The fitness values of children are compared with those of parents and in case the fitness values of children are better than those of parents, the children will replace parents. Then, the fitness value of each child will be compared with the memory of that particle (pbest). If the fitness value is better than pbest, it will replace the memory of that particle. Also, if the mentioned conditions occur, the fitness value of particles will be compared with the gbest value. In case the pbest value of particle is less than the gbest value, it will replace it. Algorithm 2 represents the pseudo code of this operator.

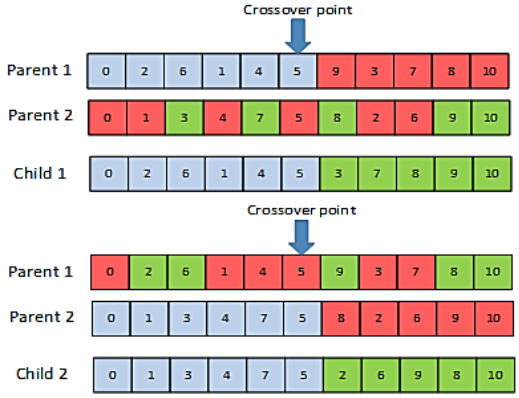

Fig. 3. Crossover operator

6.6 Mutation Operator

This operator replaces a gene with another one based on a certain probability. Mutation operator causes variety and diversity in the population. Accordingly, it expands the search space and prevents the algorithm from local optimization. Usually, this operator is done after the crossover operator and helps to gain a better solution. a new chromosome is obtained by exchanging two genes if the precedence constraint is not violated [18]. In this paper, the mutation operator is inspired from [18]. In other words, at first, a gene is randomly selected. Then, based on this method, the first successor for the task (t_j) from the mutation point to the end is obtained. If there is m^{th} gene which is a member of $[i+1, j-1]$ and the priorities of t_m are not in front of t_i, t_i and t_j can be replaced with each other which is illustrated in Figure 4. If these conditions do not occur, hence, the mutation operator will be executed from the beginning. After exchanging the gens, the fitness of the child is calculating by fitness function. The fitness value of the generated child will be compared with its parent. If the fitness results of the child is better, the child will replace with the parent. After that, the fitness value of the child will be compared with the memory of that particle (pbest). The fitness value of the particle is replaced with pbest if the obtained fitness betters the pbest. Moreover, if this condition is established, the fitness value of the particle will be compared with the gbest value and in case the value of this particle is less than the gbest value, it will be replaced with the gbest. Algorithm 3 represents the pseudo code of the mutation operator.

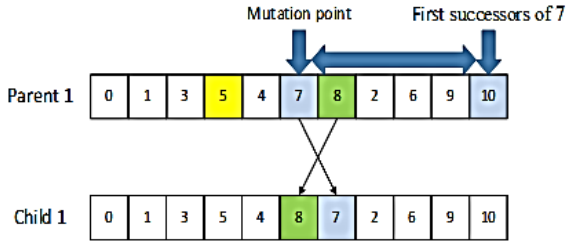

Fig. 4. Mutation operator

Algorithm 3. Pseudo code of two-point mutation operator

1: A randomly chosen chromosome.
2: Choose randomly a gene T_i in the selected chromosome;
3: Find the first successor $T_j \in Succ(i)$;
4: Choose randomly a gene T_k in the interval $[i+1, j-1]$;
5: **if** l < i for all $T_l \in Pred(k)$ **then**
6: Generate a new offspring by interchanging gene T_i and gene T_k;
7: **return** the new offspring;
8: **else**
9: Go to Step 1;
10: **if** the fitness of the new offspring is better than its parent **then** replace it with parent
11: **if** step 10 is true **then** compare fitness value of offspring with local best **if** offspring's fitness value is better **then** update local best.
12: **if** step 11 is true **then** compare fitness value of offspring with global best **if** offspring's fitness value is better **then** update global best.

6.7 Termination Condition

The genetic and PSO algorithms are regarded as random methods which can be executed for ever by means of a rule. In practice, a termination condition should be carried out. The usual methods operate by considering the fitness evaluations or the working times of the computer or by exploring the population diversity. In this paper, the termination condition is realized when the algorithm has been executed for 1000 times.

6.8 Complexity Analysis

The complexity of the proposed method is *O (geners* $\times n^2 \times e \times m$*)*, where *geners* is the number of iterations, *n* is the number of subtasks, *e* is the number of edges and *m* is the number of machines.

7. Experimental Results

Certain measurement criteria were used for evaluating efficiency which are mentioned later in the paper. It should be noted that the entire implementation procedure was conducted in Visual Studio 2013 and the C#.net programming language was used to implement the algorithm. There are some parameters in the combined algorithm which have a significant impact on the performance of the algorithm; these parameters are given in Table 3. In this table, the parameter *ini* determines the

number of population (particles) and the parameter w stands for the inertia weight which is aimed at balancing the speed of particles. The values of the parameters C_1 and C_2 help particles learn how to locate the optimal points. The parameters srate, crate, mrate refer to the rates of the selection, crossover and mutation operators, respectively.

Table 3. Values of the parameters

parameters	Values
ini	80
C_1	1.5
C_2	1.5
r_1	Randomly
r_2	Randomly
W	0.4
srate	30
crate	80
mrate	20

7.1 Comparing Measurements

In this section the proposed algorithm is compared with other three heuristics and a GA algorithms in term of the makespan. To do this, some metrics such as SLR and CCR are used in comparison. Furthermore, random graph and statistical analysis are used in experimental comparisons.

7.1.1 Task Makespan

The makespan of the directed acyclic graph, as shown in Figure 1, was simulated on three prioritization methods by means of upward rank, downward rank and the combination these two methods and MPQMA algorithm. The simulation results for Figure 1 with the proposed and other algorithms are illustrated in Figure 6, Figure 7, Figure 8 and Figure 9 respectively. The obtained results as in mentioned figures were 76, 74, 76 and 70 for the four algorithm. When scheduling by the proposed algorithm is finished, the makespan for graph execution is equal to 68 which is illustrated in Figure 10. These gained results from algorithms are shown in Figure 5 by Bar chart which depicts that the makespan of scheduling for the proposed algorithm has better performance than the other four algorithms.

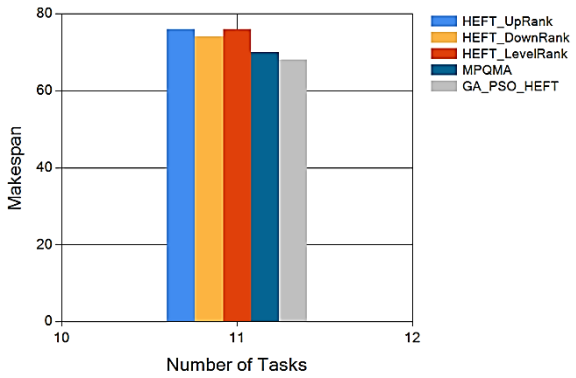

Fig. 5. Bar graph representing the makespan vs. number of tasks for the graph of Fig. 1

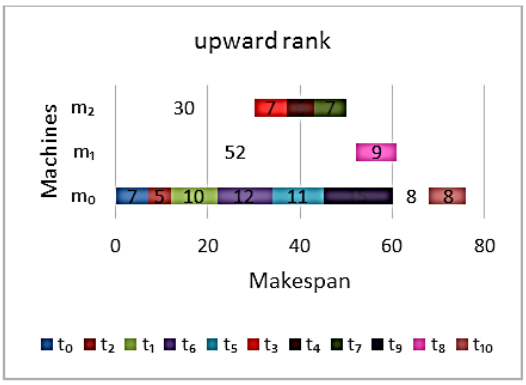

Fig. 6. Gantt chart for task scheduling with upward rank prioritization (Makespan = 76)

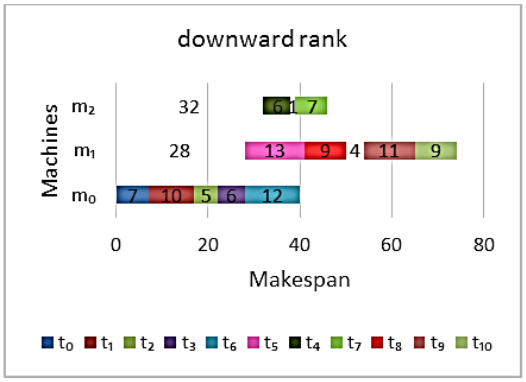

Fig. 7. Gantt chart for task scheduling with downward rank prioritization (Makespan = 74)

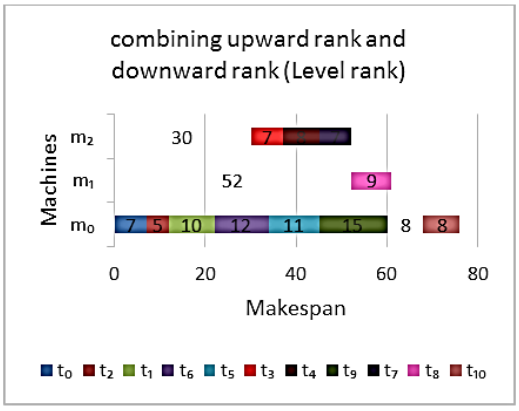

Fig. 8. Gantt chart for task scheduling with both upward and downward ranks (Makespan = 76)

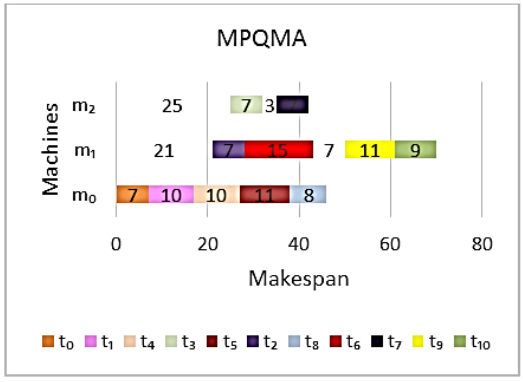

Fig. 9. Gantt chart for task scheduling with MPQMA (Makespan = 70)

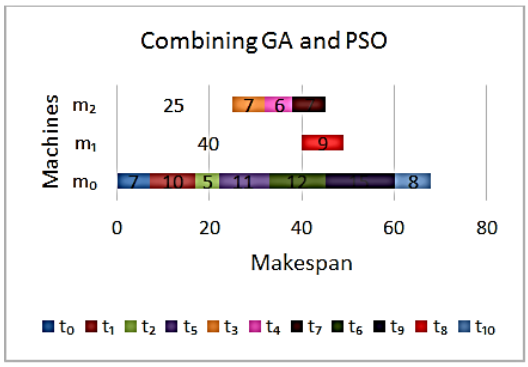

Fig. 10. Gantt chart for task scheduling with a combination of the prioritization of genetic and PSO algorithms (Makespan = 68)

7.1.2 Scheduling Length Ration (SLR)

In this paper, for measuring SLR, the minimum critical path between the processors or machines should be obtained. Hence, the critical path method (CPM) was used [28].

The measurement of the main efficiency of the scheduling algorithm on the graph is the scheduling length. Since a large collection of task graphs with different features is used, the scheduling length to the low bound should be converted into a rule which is referred to as scheduling length ratio (SLR). The value of SLR on the graph is obtained through Equation (15).

The denominator of the ratio is the set of minimum computation costs of the tasks on CP_{min}. In an unscheduled directed acyclic graph, if the computation cost of each node n_i is adjusted with less value, then, the critical path will be based on minimum computation costs which are indicated by CP_{min}. The SLR of a graph cannot be less than one. In the task scheduling algorithm, the smallest SLR will have better efficiency [18].

$$SLR = \frac{makespan}{\sum_{T_i \in CP_{min}} \min_{P_k \in P} \left(\omega(T_i, P_k)\right)} \quad (15)$$

7.1.3 Communication to Computation Ratio (CCR)

CCR indicates that the used DAGs in this paper is either communication-intensive or computation-intensive. For a graph, the value of CCR is obtained by measuring the mean communication cost (numerator of Equation (16)) divided by the mean computation cost (denominator of Equation (16)) in the computational system. Hence, CCR value is measured through the following equation [18].

$$CCR = \frac{\frac{1}{e}\sum_{edge(T_i,T_j)\in E} \overline{C(T_i,T_j)}}{\frac{1}{n}\sum_{T_i \in T} \overline{\omega(T_i)}} \quad (16)$$

7.2 Comparison of SLR vs. CCR for Graph Depicted in Fig. 1

The comparison of SLR vs. CCR calculations in the proposed method was done and simulated using upward and downward ranks and using the combination of priority methods. The results of comparisons are depicted in Figure 11. As shown in this figure, the method proposed in this paper has less scheduling length rate which is attributed to the fact that the makespan in the proposed method is less than other methods. Consequently, this leads to the minimization of Equation (15).

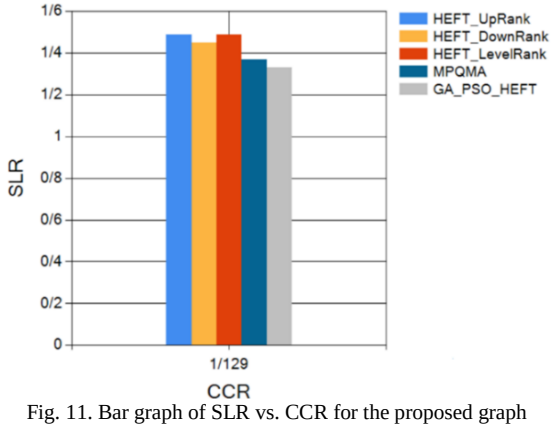

Fig. 11. Bar graph of SLR vs. CCR for the proposed graph

7.3 Comparing SLR vs. CCR for Random graph

In this paper, for a more extensive comparison and evaluation, randomly produced graphs were used. In this section, a random graph is examined. The directed acyclic graph which was randomly produced has 30 tasks; in total, it has 72 edges. This graph was executed on 9 machines. As illustrated in Figure 12, the proposed method has less SLR than the other methods. Furthermore, in the proposed method, the makespan on the random graph was 176. Accordingly, with respect to the results demonstrated in Figure 13, it can be maintained that the proposed method has better performance than the other methods.

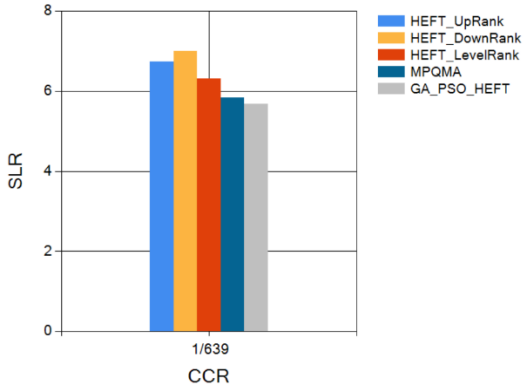

Fig. 12. Bar graph of SLR vs. CCR for the random graph

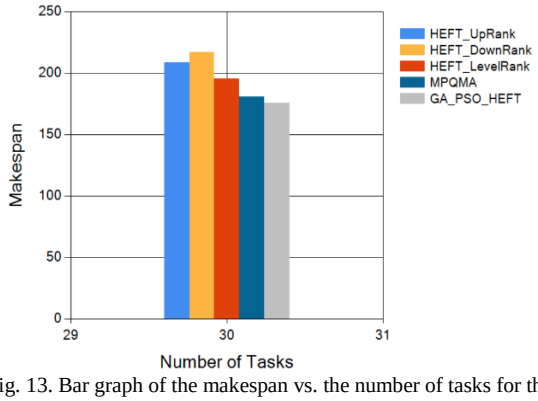

Fig. 13. Bar graph of the makespan vs. the number of tasks for the random graph

7.4 Evaluation of the Randomly Produced Graphs

For evaluating results on different graphs with 10, 50 and 100 tasks using 8 machines, 100 iterations of the three mentioned tasks were produced. Figure 14, Figure 15 and Figure 16 represent the results obtained from the experiments. In Figure 14, average makespans of 100 independent task graphs with 10 tasks for HEFT_UpRank, HEFT_DownRank, HEFT_LevelRank, MPQMA and the proposed algorithms are: 99.27, 101.03, 99.48, 92.64 and 92.18 respectively. With respect to the obtained results the proposed algorithm has better performance than the other methods.

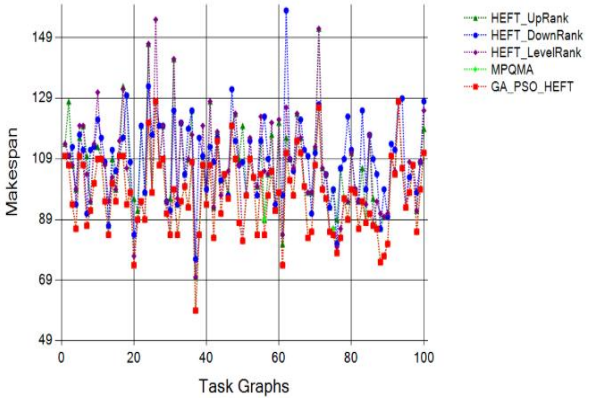

Fig. 14. Makespan of the produced 10-fold graph with 8 machines and 100 independent executions

According to the results in Figure 15, the average makespans of 100 independent task graphs scheduling with 50 tasks for the HEFT_UpRank, HEFT_DownRank, HEFT_LevelRank, MPQMA and the proposed algorithms are 469.29, 487.41, 469.69, 461.54 and 435.97 respectively. With respect to the obtained results the proposed algorithm betters makespans of the other methods.

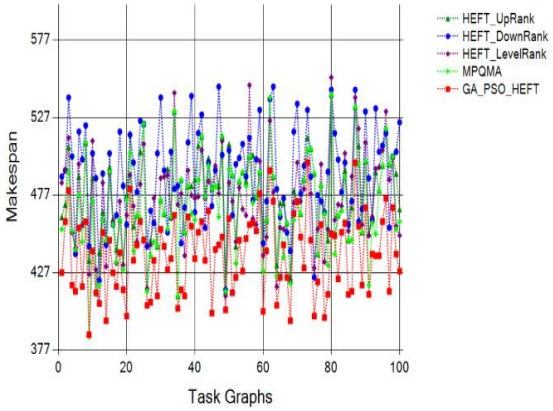

Fig. 15. Makespan of the produced 50-fold graph with 8 machines and 100 independent executions

Furthermore, the obtaining average makespans results of 100 independent task graphs with 100 tasks in Figure 16 for HEFT_UpRank, HEFT_DownRank, HEFT_LevelRank, MPQMA and the proposed algorithms are: 948.79, 985.41, 946.52, 939.73 and 897.3 respectively. According to the results the proposed algorithm optimizes the other methods in terms of makespan.

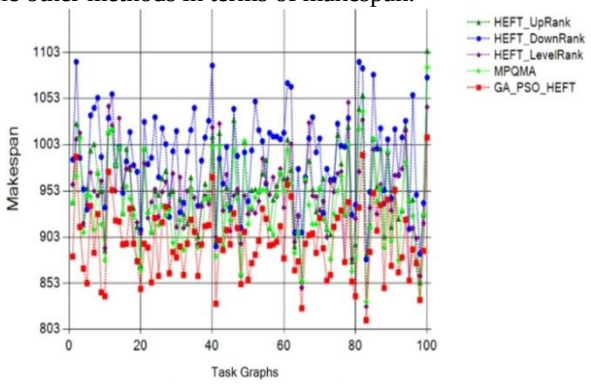

Fig. 16. Makespan of the produced 100-fold graph with 8 machines and 100 independent executions

8. Conclusion and Suggestions for Further Research

As discussed in the paper, task scheduling is considered to be one of critical challenges in cloud computing systems. In the past, numerous task scheduling methods have been used in cloud computing. In this paper, to enhance resource efficiency and minimize the total task execution time, the researchers used a novel cost function which was based on a combination of PSO and genetic algorithms. The cost function was used to measure task execution time on available resources in the context of cloud computing. The purpose of proposing the hybrid or combinatory model was to benefit from the capabilities meta-heuristic methods since they have high speed in finding optimal solutions. The new method introduced in the proposed algorithm was intended to reduce and shorten the length of the critical path and reduce the communication costs among the processors. Finally, the obtained results from the implementation of the proposed method indicated that it optimizes other mentioned current algorithms. In future, it is possible to design an appropriate scheduling for similar algorithms.

References

[1] A. Ghaffari, "Real-time routing algorithm for mobile ad hoc networks using reinforcement learning and heuristic algorithms," Wireless Networks, pp. 1-12, 2016.

[2] Z. Mottaghinia and A. Ghaffari, "A Unicast Tree-Based Data Gathering Protocol for Delay Tolerant Mobile Sensor Networks," Information Systems & Telecommunication, p. 59, 2016.

[3] A. Ghaffari, "Congestion control mechanisms in wireless sensor networks: A survey," Journal of Network and Computer Applications, vol. 52, pp. 101-115, 6// 2015.

[4] C.-S. Chen, W.-Y. Liang, and H.-Y. Hsu, "A cloud computing platform for ERP applications," Applied Soft Computing, vol. 27, pp. 127-136, 2// 2015.

[5] Y.-D. Lin, M.-T. Thai, C.-C. Wang, and Y.-C. Lai, "Two-tier project and job scheduling for SaaS cloud service providers," Journal of Network and Computer Applications, vol. 52, pp. 26-36, 6// 2015.

[6] R. Masoudi and A. Ghaffari, "Software defined networks: A survey," Journal of Network and Computer Applications, vol. 67, pp. 1-25, 5// 2016.

[7] M. Pinedo, Scheduling : theory, algorithms, and systems, 4th ed. New York: Springer, 2012.

[8] Y. Robert and F. d. r. Vivien, Introduction to scheduling. Boca Raton: CRC Press, 2010.

[9] F. Magoulès, J. Pan, and F. Teng, Cloud Computing : Data-Intensive Computing and Scheduling. Boca Raton: CRC Press, 2012.

[10] Y. Li and W. Cai, "Update schedules for improving consistency in multi-server distributed virtual environments," Journal of Network and Computer Applications, vol. 41, pp. 263-273, 5// 2014.

[11] B. Xu, C. Zhao, E. Hu, and B. Hu, "Job scheduling algorithm based on Berger model in cloud environment," Advances in Engineering Software, vol. 42, pp. 419-425, 7// 2011.

[12] T. S. Somasundaram and K. Govindarajan, "CLOUDRB: A framework for scheduling and managing High-Performance Computing (HPC) applications in science cloud," Future Generation Computer Systems, vol. 34, pp. 47-65, 5// 2014.

[13] S. Su, J. Li, Q. Huang, X. Huang, K. Shuang, and J. Wang, "Cost-efficient task scheduling for executing large programs in the cloud," Parallel Computing, vol. 39, pp. 177-188, 4// 2013.

[14] B. Keshanchi and N. J. Navimipour, "Priority-Based Task Scheduling in the Cloud Systems Using a Memetic Algorithm," Journal of Circuits, Systems and Computers, vol. 25, p. 1650119, 2016.

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

[16] Y. C. Liang, A. H. L. Chen, and Y. H. Nien, "Artificial Bee Colony for workflow scheduling," in 2014 IEEE Congress on Evolutionary Computation (CEC), 2014, pp. 558-564.

[17] L. Wang and L. Ai, "Task Scheduling Policy Based on Ant Colony Optimization in Cloud Computing Environment," in LISS 2012: Proceedings of 2nd International Conference on Logistics, Informatics and Service Science, Z. Zhang, R. Zhang, and J. Zhang, Eds., ed Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 953-957.

[18] Y. Xu, K. Li, J. Hu, and K. Li, "A genetic algorithm for task scheduling on heterogeneous computing systems using multiple priority queues," Information Sciences, vol. 270, pp. 255-287, 6/20/ 2014.

[19] X. Kong, C. Lin, Y. Jiang, W. Yan, and X. Chu, "Efficient dynamic task scheduling in virtualized data centers with fuzzy prediction," Journal of Network and Computer Applications, vol. 34, pp. 1068-1077, 7// 2011.

[20] H. Topcuoglu, S. Hariri, and W. Min-You, "Performance-effective and low-complexity task scheduling for heterogeneous computing," Parallel and Distributed Systems, IEEE Transactions on, vol. 13, pp. 260-274, 2002.

[21] G. Giftson Samuel and C. Christober Asir Rajan, "Hybrid: Particle Swarm Optimization–Genetic Algorithm and Particle Swarm Optimization–Shuffled Frog Leaping Algorithm for long-term generator maintenance scheduling," International Journal of Electrical Power & Energy Systems, vol. 65, pp. 432-442, 2// 2015.

[22] R. L. Haupt and S. E. Haupt, Practical genetic algorithms, 2nd ed. Hoboken, N.J.: John Wiley, 2004.

[23] F. T. Hecker, M. Stanke, T. Becker, and B. Hitzmann, "Application of a modified GA, ACO and a random search procedure to solve the production scheduling of a case study bakery," Expert Systems with Applications, vol. 41, pp. 5882-5891, 10/1/ 2014.

[24] S. N. Sivanandam and S. N. Deepa, Introduction to genetic algorithms. Berlin ; New York: Springer, 2007.

[25] A. Mahor and S. Rangnekar, "Short term generation scheduling of cascaded hydro electric system using novel self adaptive inertia weight PSO," International Journal of Electrical Power & Energy Systems, vol. 34, pp. 1-9, 1// 2012.

[26] D. Y. Sha and H.-H. Lin, "A multi-objective PSO for job-shop scheduling problems," Expert Systems with Applications, vol. 37, pp. 1065-1070, 3// 2010.

[27] A. P. Engelbrecht, Computational intelligence : an introduction, 2nd ed. Chichester, England ; Hoboken, NJ: John Wiley & Sons, 2007.

[28] U. Defense Acquisition and Press, Scheduling guide for program managers. Fort Belvoir, VA; Washington, DC: Defense Acquisition University Press ; For sale by the U.S. G.P.O., Supt. of Docs., 2001.

Amin Kamalinia received his BS degree in Software Engineering from Bostan-Abad Branch, Islamic Azad University, Bostan-Abad, Iran, in 2011 and his MS degree in Software Engineering from Science and Research Branch, Islamic Azad University, Urmia, Iran in 2014. His research interests include Grid & Cloud computing, Task Scheduling and Programming.

Ali Ghaffari received his BSc, MSc and PhD degrees in computer engineering from the University of Tehran and IAUT (Islamic Azad University), TEHRAN, IRAN in 1994, 2002 and 2011 respectively. As an assistant professor of computer engineering at Islamic Azad University, Tabriz branch, IRAN, his research interests are mainly in the field of wired and wireless networks, Wireless Sensor Networks (WSNs), Mobile Ad Hoc Networks(MANETs),Vehicular Ad Hoc Networks(VANETs), networks security and Quality of Service (QoS). He has published more than 60 international conference and reviewed journal papers.