An Effective Task Scheduling Framework for Cloud Computing using NSGA-II

Hanieh Ghorashi[1], Meghdad Mirabi[2]
1- Department of Computer Engineering, Islamic Azad University, Central Tehran Branch, Tehran, IRAN. (hanieh.ghorashi@gmail.com)
2- Department of Computer Engineering, Faculty of Engineering, Islamic Azad University, South Tehran Branch, Tehran, IRAN.

Received (2020-01-09) Accepted (2020-07-15)

Abstract: *Cloud computing is a model for convenient on-demand user's access to changeable and configurable computing resources such as networks, servers, storage, applications, and services with minimal management of resources and service provider interaction. Task scheduling is regarded as a fundamental issue in cloud computing which aims at distributing the load on the different resources of a distributed system in order to optimize resource utilization and response time. In this paper, an optimization-based method for task scheduling is presented in order to improve the efficiency of cloud computing. In the proposed approach, three criteria for scheduling, including the task execution time, the task transfer time, and the cost of task execution have been considered. Our method not only reduces the execution time of the overall tasks but also minimizes the maximum time required for task execution. We employ the Multi-objective Non-dominated Sorting Genetic Algorithm (NSGA-II) for solving the scheduling problem. To evaluate the efficiency of the proposed method, a real cloud environment is simulated, and a similar method based on Multi-Objective Particle Swarm Optimization is applied. Experimental results show the superiority of our approach over the baseline technique.*

Keywords: *Cloud Computing, Task Scheduling, Multi-Objective Optimization, NSGA-II, Load Balancing.*

How to cite this article:
Hanieh Ghorashi, Meghdad Mirabi. An Effective Task Scheduling Framework for Cloud Computing using NSGA-II. J. ADV COMP ENG TECHNOL, 6(3) Summer 2020 : 155-168

I. INTRODUCTION

Cloud computing is a model for convenient on-demand user's access to changeable and configurable computing resources such as networks, servers, storage, applications, and services with minimal management of resources and service provider interaction [1]. The cloud environment consists of a group of servers that organize various resources and provide secure, reliable, fast and transparent services to the user. One of the most important problems in cloud computing is load balancing [2]. Load balancing means distributing loads between different nodes in a distributed system to improve resource utilization and response time. Load balancing techniques help to distribute the load on all nodes equally. In the absence of load balancing, a node may have many tasks and other nodes are idle. Load balancing results in high utilization of resources and user satisfaction, which in turn serves both cloud providers and users. The goals of load balancing include increasing efficiency, having a backup program in cases where the whole or part of the system fails, maintaining system stability, implementing future modifications in the system, and reducing energy consumption to name a few [3]. Load balancing can be considered as a four-stage decision-making process. These stages contain determining the task migration time, determining the location of the task migration, determining which processor requests the migration, and specifying the tasks to be migrated.

Task scheduling algorithms in distributed systems generally intend to distribute the load on processors and maximize their utilization while reducing the tasks execution time [4]. In the cloud environment, the number of tasks, as well as the number of available resources can grow rapidly, especially when virtual resources are allocated. Calculating all possible task-resource mappings in the cloud environment, and then selecting the optimal mapping is not feasible because complexity grows exponentially with the number of tasks and resources. Fortunately, meta-heuristic algorithms ensures the efficient implementation of the scheduling algorithm, as they significantly reduce/limit the search space. There are several meta-heuristic algorithms such as genetic algorithm [5], genetic-fuzzy algorithm [6], multi-objective genetic algorithm [7], particle swarm optimization [8] and ant colony optimization [9] for optimizing the load balancing. Generally, they follow two main objectives: 1) Minimizing the task execution time, and 2) Minimizing the task execution cost in the cloud environment.

The majority of the research works in this domain assume that the objective functions in a multi-objective task scheduling model are not in conflict with each other and they have the same tendency. Therefore, authors have used single-objective meta-heuristic algorithms to solve their optimization problem [10]. However, this is a not a completely valid assumption. Different objectives have various ranges for cost, which utilizing a single-objective algorithm for minimizing the overall cost may not consider all objectives to an equal extent. In this case, a possible solution can be optimum only regarding some dominant objectives. Besides, the existing optimization frameworks do not take into account all necessary aspects of the load balancing problem simultaneously. This may result in a task assignment that considers some perspectives such as task execution time while disregards other aspects like task transfer time.

In this paper, we propose a method for optimizing the task scheduling to balance the load and improve the cloud productivity. The devised approach considers three factors including task execution time, task transfer time, and cost of the task execution in order to achieve the optimal task-resource mapping. In order to minimize the total task completion time, not only the total tasks execution time is reduced, but also the maximum total task execution time is minimized. Therefore, our method covers all necessary aspects of the scheduling problem to generate a solution that fulfills the entire objective set. We employ Non-dominated Sorting Genetic Algorithm II (NSGA-II) for solving the optimization problem. There are several reasons for this choice. Compared to Multi-Objective Particle Swarm Optimization (MOPSO), it has the mutation operator which prevents the algorithm to stick in the local optimum points. Moreover, the crowding distance operator in NSGA-II preserves the population diversity in the different generations which enhances the exploration of the algorithm, and accordingly increases the chance for finding the optimal solution. An effective multi-objective algorithm like NSGA-II considers every aspects of the optimization problem equally and produces a solution that is optimum with respect to each objective function.

The rest of the paper is organized as follows. In Section II we provide a review of existing load balancing techniques for cloud computing. Section III presents our proposed task scheduling framework. Simulation results and experiments are reported in Section VI. Finally, we conclude the paper in Section V.

II. RELATED WORKS

So far, many algorithms have been proposed for load balancing in the cloud computing. In the following, we provide a review of the most successful approaches:

Active Clustering Algorithm: This algorithm works by grouping similar nodes [11]. A node starts the process and chooses a node different from the previous node among its valid neighbor nodes as the intermediate node. Then, the intermediate node creates a connection with the neighbor of the initial node type. Then the intermediate node interrupts the connection with the initial node. Above processes are repeated.

In this algorithm, the efficiency of the system increases with resources. As a result, throughput increases with these resources.

Bee colony Algorithm: This algorithm examines the load balancing in web servers based on increasing or decreasing demand [12]. Allocated servers dynamically customize user requests. These servers are grouped as virtual servers. Each virtual server has its own virtual queues. Each server calculates required benefits by processing a request from the queue. The amount of these benefits equals to the time consumed by the processor to process a request. The Bee dance section is similar to the bulletin board. It also uses this plane to declare the benefits of the entire colony. Each of these servers plays the role of a tracer or a watch. After processing a request, the server can send benefits to notice boards with the probability of rp. A server can select a queue from virtual servers with the probability of xp (xp represents the behavior of the tracer), or it can check the notifications (watching dance) and serve which in that case, demonstrates the behavior of the Scout. In order to serve a request, after calculating the benefits and comparing it with the total colony benefit, a server considers the value of xp for it. If these benefits were high, the server would remain on the same current virtual server and a notification is sent with a probability of rp. If the benefits are low, the server will take the role of scout or tracer. This method achieves the general load balancing through the activities of local servers. As system diversity increases, system performance increases. But operating power does not increase as system size increases. This algorithm is suitable where a variety of services are required.

Random Sampling: In this algorithm, a virtual graph is constructed in which each node (server as a node) has a connection to represent the server load [13]. In this graph, each server is directed to server free resources as a node of any degree. Whenever a node performs a task, that node is deleted from the input edges, which indicates that the access to resources is reduced and that resource is freed. After completing a task, the node creates an input edge, which indicates increased access to the free resource. Adding and deleting the process is conducted by random sampling. A neighbor is randomly selected at the beginning of the path in each node. The last node is selected for load allocation. There are also other ways to select the node for load allocation and specific criteria are used to select the node. However, the selection and allocation of load can be for low-load nodes. Finally, a directed graph is obtained. This method balances load across all system nodes. System efficiency increases with increasing the number of resources, resulting in increased throughput by increasing system resources. Here, the load balancing plan is completely decentralized, so it is suitable for large network systems such as cloud.

CARTON method: In order to control the application of load balancing and Distributed Rate Limiting (DRL), CARTON mechanism has been proposed for the cloud [14]. Load balancing is used to distribute tasks to different servers, so the corresponding costs can be reduced. Distributed Rate Limiting (DRL) is used to ensure that resources are distributed in a way that fairly distribution is preserved. This algorithm can be easily implemented with very low computing and communication overhead. A unified framework is where this algorithm is applied for cloud control. In this algorithm, the overhead and resources utilization are considered among the load balancing criteria.

Event-driven method: An event-driven load balancing algorithm is proposed for multiplayer online games [15]. After receiving capacity events as input, this algorithm analyzes its components in terms of the resources and the general state of the game's session. Consequently, load balancing activities generate a game session. This method is capable of reducing or increasing the scale of a game session on multiple resources based on the user variable load. However, there is an occasional lack of quality of service. In this algorithm, the resource utilization is considered as one of the load balancing criteria.

Server-based Load Balancing for Internet Distributed Services: A new service based load balancing policy is proposed for web servers distributed across the world [16]. This policy helps reduce service times by limiting the number of paths to a request to the closest remote server without overloading them. A middleware is described for implementation of this protocol. Also, in order to withstand the load, it uses a metaheuristic method to help web servers.

Fuzzy logic: A cyclic turn-based load balancing algorithm is designed in virtual machine environments in cloud computing in order to achieve better response time and processing time [17]. The load balancing algorithm runs before the processing servers arrive. The task is scheduled based on various parameters such as processor speed and load allocated to the virtual machine. This algorithm keeps the information inside each virtual machine and the number of current requests allocated to the virtual machine of the system. When there is a task for allocation, this algorithm specifies a machine with a minimum load, and if there is more than one machine with this property, this algorithm determines the first machine. Researchers have tried to implement a new load balancing technique based on fuzzy logic. Since fuzzy logic is similar to natural language, it can formulate its own problems. In this architecture, the fuzzifier executes the process of fuzzification, which receives two types of input data, such as processor speed and load allocated to the virtual machine, and sends an output that is the same balanced load. This design takes into account the processor speed and virtual machine load as two input parameters in order to better balance the load in the cloud using fuzzy logic. These parameters are given as inputs to the fuzzifier mechanism, which are used to measure the balanced load as output. The two parameters of the processor speed and virtual machine allocated load are used together to evaluate the balanced load on cloud computing data centers through fuzzy logic. The results obtained by evaluating efficiency can achieve load balances by reducing processing time and improving response time, which results in maximum resource utilization. The processor speed and load allocated to the virtual machine for load balancing in cloud computing is applied through fuzzy logic.

Message-based Model: In this model, clusters provide the opportunity to use applications distributed by different computers across networks [18]. This issue is related to the clusters in the network performance, if the total load on the distribution network is distributed by a computer, it will slow down the network. In order to avoid this situation, resource management can use software criteria to distribute traffic between stations, so that network performance is preserved in a high probability. Web services are mainly used in instant messaging applications, which is a technology for real-time communications between different parties. However, the application's availability is important. In this algorithm, the response time and efficiency are considered as two load balancing criteria.

Min-Min Algorithm: This algorithm [19] starts with a set of unallocated tasks. First of all, the minimum total completion time is found. Then, the minimum value is selected among these minimum times which is the minimum time per resource. Then, according to the minimum time, task is scheduled on the corresponding machine. Afterwards, the runtime for all other tasks on the machine is updated by adding the running time of the allocated task to the execution time of the other tasks on the machine, and the allocated task is removed from the list of tasks allocated to the machine. The same procedure is followed up until all tasks are allocated to the resources. But, this has a major problem, which it can lead to starving. In this algorithm, the resource utilization, overhead, throughput, response time, and efficiency are considered as the load balancing criteria.

Min-Max Algorithm: The Min-Max algorithm is almost the same as the Min-Min algorithm [20], except for the following: the maximum value is selected after finding minimal run times which is the maximum time per resource. Afterwards, according to the maximum time, the task is scheduled on the corresponding machine. Then, the runtime for all other tasks on the machine will be updated by adding the execution time of allocated task to the execution time of the other tasks on the machine and the allocated task is deleted from the list of tasks allocated to the machine. In this algorithm, as in the Min-Min algorithm, resource utilization, overhead, throughput, response time, and efficiency are considered as load balancing criteria.

OLB + LBMM Two-stage load balancing algorithms: A two-stage scheduling algorithm has been suggested that the scheduling algorithms (OLB load balancing opportunity) and (LBMM load balancing) are used for better performance and maintaining the system load balancing [21]. The OLB scheduling algorithm keeps each node working in order to achieve the load balancing, and the LBMM scheduling algorithm is used to reduce the runtime of each task on the node, thus reducing the total runtime. The three-level cloud computing networks are where this algorithm is used, in which the efficiency and resource utilization criteria are considered. This combined algorithm helps for efficient resource utilization and increases throughput. This algorithm offers better results than Bee colony, random sampling, and active clustering algorithms.

Queue-Idle-Join algorithm: This algorithm provides a large-scale load balancing with distributed distributors [22]. First, the load balancing is conducted to make each idle processor have access to any distributor, and then allocate tasks to the processors in order to reduce the average length of the queue of each processor. After task deletion from the critical paths of processing requests, this algorithm effectively reduces system load. No communication overhead occurs when the tasks arrive and does not increase the actual response time. Cloud data center is where this algorithm is used in which the efficiency, response time, and overhead criteria are considered.

Central load balancing policy for virtual machines: In this method, a central load balancing policy is proposed for virtual machines that balances load equally in a cloud computing or distributed virtual machines [23]. This policy increases overall system performance but does not consider systems that have an error tolerance. This method uses general state information for load balancing decisions and increases efficiency by more than 20%. Cloud computing is where this algorithm is used in which the efficiency and response time, throughput, and resource utilization are considered.

Distributed Scheduling Hill Climbing (DSHC): This paper proposes a dynamic scheduling algorithm that uses hill climbing algorithm [24]. It tries to minimize completion time of tasks while maximizing throughput and utilization of resources. This algorithm allocates independent tasks to available resources to achieve load balance. The simulation results show that the algorithm can achieve load balance and reduces completion time of tasks.

III. PROPOSED METHOD

In general, task scheduling is considered as a solution to the load balancing problem in cloud computing. The most important issue in task scheduling is the formulation of scheduling objectives and the model being tested. More precisely, we must define equations that cover all the aspects of a scheduling problem and identify their goals well. Given that finding optimal mapping is subjected to the minimization of several different objectives, in this paper, the multi-objective NSGA-II is applied in order to find a solution that results in the minimal value for the three cost functions mentioned above. In the following, the formulation of three objective functions is defined, and details of the optimization algorithm are then presented. Finally, the routine and flowchart of the proposed method are expressed.

In order to formulate cost functions, the following variables and constraints are used:

- n: number of tasks
- $T=\{t_1, t_2, \ldots, t_n\}$: The set of tasks in the waiting queue
- N_{PM}: The number of physical machines in the cloud
- m: The number of virtual machines
- VM_j: For a set of m virtual machines, VM_j is the same as the *jth* virtual machine.
- $VM_z = \{VM \in$ z'th PM, $z \in \{1, 2, \ldots, N_{PM}\}\}$: The set of VMs allocated to the z'th PM.
- $VMp = \{VM|VM$ is allocated to p'th Cloud provider, $P \in \{1, 2, \ldots, cp\}\}$: The VMs that are allocated to the P'th cloud provider.
- B_{CK}: Bandwidth between center and V_{Mk}
- cp: The number of cloud providers
- $\hat{C}_p$: The maximum capacity for the p'th service provider in the cloud

- DE_{ik}: The amount of data allocated by task i to VMk
- VMm_k: VM_k memory size
- VMc_k: VM_k Capacity
- $Pcost_j$: The cost of a VM unit for the j'th provider (one dollar per hour)
- r_p: The total number of VMs provided by the k provider which performs the tasks at the time interval pt.
- $x_{ik} \in \{0,1\}, \quad \forall i = 1, \dots, n \;\; \& \;\; k = 1, \dots, m$
- $0 \le r_p \le \tilde{C}_p, \quad \forall p = 1,2,\dots,cp$

It is noteworthy that in the proposed method if task i is allocated to VM_k, x_{ik}=1, otherwise x_{ik}=0.

The equation (1) to calculate the task execution time on a virtual machine k is used in the proposed method:

$$Texe_k = \sum_{i=1}^{n} x_{ik} \times \frac{DE_{ik}}{VMm_k \times VMc_k} \qquad (1)$$

Where $Texe_k$ refers to task execution time on VM_k.

Using equation (1), the total execution time is obtained by equation (2):

$$Texe = \sum_{k=1}^{m} Texe_k \qquad (2)$$

Also, in the proposed method, the task transfer time can be obtained using equation (3):

$$Ttrans = \sum_{k=1}^{m} \sum_{i=1}^{n} x_{ik} \times \frac{DE_{ik}}{B_{ck}} \qquad (3)$$

The total execution cost for the cloud provider (one dollar per hour) is obtained using equation (4).

$$Cexe = \sum_{p=1}^{cp} \left(Pcost_p \times r_p \times \left(\sum_{k \in SP_p} Texe_k \right) \right) \qquad (4)$$

Where r_p is determined by equation (5):

$$r_p = \sum_{k \in SP_p} \min \left(\sum_{i=1}^{n} x_{ik}, 1 \right) \qquad (5)$$

After modeling the task scheduling problem, objectives must be specified. The desired objectives are runtime, transfer time, cloud provider cost, as defined by (6) - (8):

$$\min f(ExecutionTime) = Texe + \left(\max_{1 \le i \le k} Texe_i \right) \qquad (6)$$

$$\min f(TransferTime) = Ttrans \qquad (7)$$

$$\min f(Cost) = Cexe \qquad (8)$$

Optimization problems that have more than one objective function are common in many fields. In such cases, the objective functions naturally contradict each other. This means that there is no single solution for such issues. Therefore, the goal is to find solutions that make consistency and consensus between the objectives. A multi-objective optimization problem is expressed in terms of equation (9):

$$\min \vec{F}(\vec{X}) = [f_1(\vec{X}), f_2(\vec{X}), \dots, f_k(\vec{X})] \qquad (9)$$

Where $\vec{X} = (x_1, x_2, \dots, x_k)$ is the vector of decision variables; $f_i: R^n \to R, i = 1, \dots, k$ are objective functions. Assuming that the decision variable $\vec{X} = (x_1, x_2, \dots, x_k)$ represents a solution. The solution $\vec{X}_2$ dominates the solution $\vec{X}_1$ if $f_j(\vec{X}_1) \ge f_j(\vec{X}_2)$ for all $j = 1, \dots, k$, and $f_j(\vec{X}_1) \geq f_j(\vec{X}_2)$ for at least one $j = 1, \ldots, k$. An acceptable solution $\vec{X}_1$ is a Pareto optimal (non-dominant) solution if there is no other acceptable solution $\vec{X}_2$ that dominates it. The set of all objective vectors of $F(\vec{X}_1)$ corresponding to Pareto optimal solutions is called the Pareto front (P*). Therefore, the objective is to determine the optimal Pareto set from the set *F* of all decision variable vectors.

By adding two essential operators to a single-objective genetic algorithm, this algorithm has been transformed into a multi-objective algorithm called Multi-objective NSGA-II [25], which a group of best solutions which is known as Pareto front instead of finding the best solution. These two operators are:

- An operator that assigns a ranking criterion based on the non-dominated sorting to members of the population.
- An operator that maintains the diversity among equal-ranking solutions.

Before the full description of this algorithm, it is necessary to explain the concept of domination, non-dominated sorting and the concept of maintenance of diversity in the solutions.

In a minimization problem with more than one objective function, X dominates Y if and only if Y is in no way better than X, and X is at least a strictly better than Y in at least one aspect. This concept is mathematically expressed by equation (10),

$$X \leq Y (X \bmod Y) \Leftrightarrow \forall i: X_i \leq Y_i \quad \wedge \quad \exists i_0: X i_0 \prec Y i_0 \tag{10}$$

When discussing a single-objective algorithm, the criterion of the superiority of solutions relative to each other is very simple and obvious. Because only one objective function is considered, and if the problem is a minimization problem, the solution that has the minimal value of the objective function is desirable and is superior to other solutions. But, when a multi-objective algorithm is used to solve a problem, it means that at least two objective functions are considered, and it is no longer possible to easily determine some of the solutions. In most cases, points are found that none is superior to the other, and cannot be compared using the concept of domination. Therefore, in order to obtain the best solutions, they should be sorted according to a certain standard. In this algorithm, a Rank is assigned to each solution, which is based on the number of dominations compared to the other points. At the end of the algorithm, the points with the best rank, that is 1, are selected as the set of solutions or points of the Pareto front. This is described in Fig. 1, which is an example of a minimization problem with two objective functions.

As shown in Fig. 1, the number of points in the space for each possible solution is specified, each of which has two values of F1 and F2 for the objective function. Here, point 3 is checked compared to other points. Point 3 is superior to all points in space A. That is, F1 and F2 for this point are less than F1 and F2 for all points on plane A. So this point always dominates the points on plane A. Also, all points in space C are superior to point 3. That is, the F1 and F2 are lower for these points than F1 and F2 for point 3. So point 3 is always dominated by points in space C. For example, point 3 dominates point 6 and is dominated by points in space A. However, it cannot be directly judged as to the position of the superiority or non-superiority of points in space B and D relative to point 3. Because the points on plane B are better than 3 in terms of F1 and worse in terms of the F2, also for the points in the space D, they are better in F2 than F1. So in this direct comparison, one cannot say which point dominates the other. In such cases, the presence of other members of the population is used to judge. First, let assume that there is no point in space C. We want to make a comparison between points 3 and point 2 in space B. As noted, each has a better situation and a worse situation. In this situation, we must see if there is another point that is better than both of these points. Point 1 is better than point 2, then point 1 dominates points 2, but there is no point that is better than point 3 in both functions. Point 2 was then dominated by other members of the population, but point 3 has never been dominated. As a result, between these two points point 3has a better situation.

For points 4 and 3, the situation is the same. That is, point 4 is dominated by point 5, but point 3 is not dominated by any point. So Point 3 is better than point 4. But for points 1 and 5, we cannot comment on point 3 because they are not dominated by any point, and each has superiority and non-superiority over each other. So points 1, 5 and 3, which have never been dominated and ranked 1, are part of the Pareto Front.

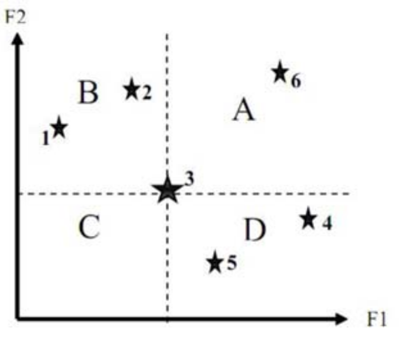

Fig. 1. Some points of the solution space of a hypothetical problem [25].

A remarkable point is that it is sometimes necessary to compare the members of a set with the same rank and some of them should be deleted. This is done using the concept of maintenance of the diversity of solutions. This means that, in deleting multiple members of a set, it tries to act in such a way that the set has a solution from each interval regularly. This is illustrated by an example.

Suppose that the points in Fig. 2 belong to the same rank set. It is necessary to remove a point from these points. Therefore, we try to select that point so that the diversity of solutions is maintained to some extent. For example, between points 3 and 5, point 3 is a better choice for deletion. Because, by deleting the point 5, there is no representative solution in the large interval of the F1 and F2 axes, respectively, that is between d and g and between n and p. But, if point 2 is selected for deletion, the solutions diversity does not diminish, because there are other solutions near this point.

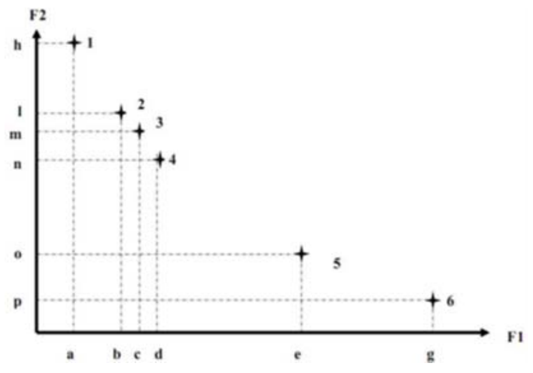

Fig. 2. The hypothetical points related to a set with an equal rank [25].

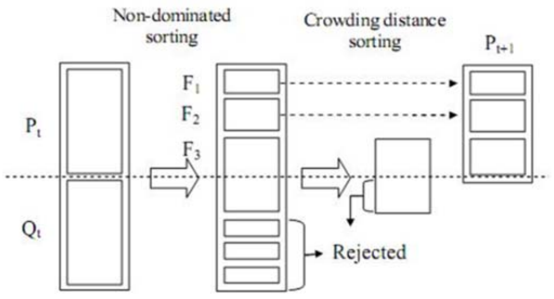

Fig. 3. Operators in the stage of solutions selection in the multi-objective NSGA-II [25]

The reason for application of these two operators in the multi-objective NSGA-II refers to the selection stage that some chromosomes should be selected from parents and children's chromosomes to begin the next stage and remove some of the solutions. This algorithm can be described in Fig. 3.

As shown in Fig. 3, the P_{t+1} members should be selected from the $P_t \cup Q_t$ members based on their rank and the rest should be eliminated. As shown in Figure 3, members with ranks 1 and 2 are all selected, but for members with a rank 3, some are deleted and the rest must be selected. As is evident, they all have the same rank and another criterion must be applied for selection, which is the same criterion for maintenance of the diversity of solutions. The operator of this stage is known as the Crowding distance. The concept of this operator is described above and its mathematical expression for point *i* in a two-objective problem is given in Fig. 4 as follows.

$$d_i^1 = \frac{\left|f_1^{i+1} - f_1^{i-1}\right|}{f_1^{max} - f_1^{min}} \qquad (11)$$

$$d_i^2 = \frac{\left|f_2^{i+1} - f_2^{i-1}\right|}{f_2^{max} - f_2^{min}} \qquad (12)$$

$$D = d_i^1 + d_i^2 \qquad (13)$$

In equations (11) - (13), the values of f_1^{i-1} ،f_1^{max} ،f_1^{min} ،f_2^{i+1} ،f_2^{i-1} ،f_2^{max} ،f_2^{min} are specified in Fig. 4, and is the ratio of the region corresponding to the domain of point *i* to the whole area of the objective function f_1, and d_i^2 is the ratio of the region to the domain of the same point to The whole region of the objective function f_2, and D, which is the sum of these two ratios, represents an index of the general domain of this point, which is called the Crowding distance. Therefore, if a point has a greater crowding distance, it will cover a greater range and its elimination will result in the loss of solution diversity across a wide range of solutions. Therefore, the points of the set of solutions with a rank of 3 that have lower crowding distance should be eliminated so that the initial population is kept constant. Also, the points of the beginning and end of this set are important points that must exist between the solutions and should not be eliminated. The above equation can be generalized for each problem with several objective functions.

As described in the multi-objective genetic algorithm, the difference between this algorithm and the single-objective genetic algorithm is in the selection stage for the parents and children in order to maintain the number of population at the beginning of each cycle [26]. Therefore, the implementation steps of a multi-objective NSGA for the task scheduling problem in cloud computing is illustrated in the flowchart of Figure 5. In the following, we describe each step in detail.

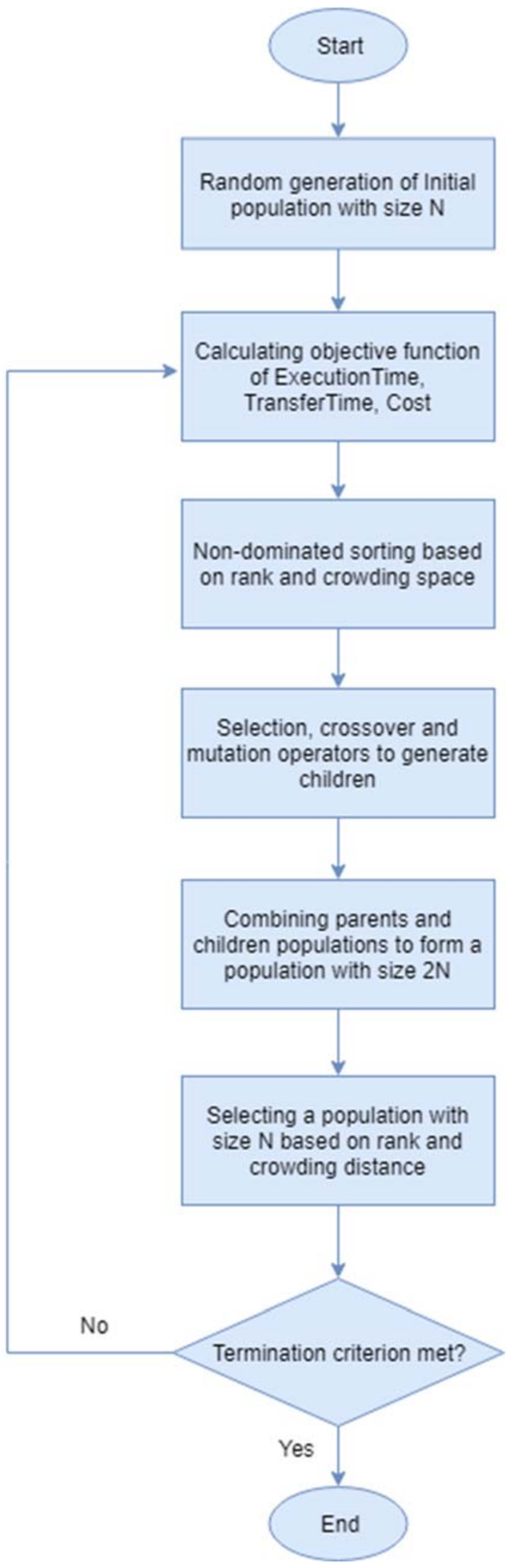
Start
Random generation of Initial population with size N
Calculating objective function of ExecutionTime, TransferTime, Cost
Non-dominated sorting based on rank and crowding space
Selection, crossover and mutation operators to generate children
Combining parents and children populations to form a population with size 2N
Selecting a population with size N based on rank and crowding distance
Termination criterion met?
No
Yes
End

Fig. 5. Multi-objective NSGA-II Flowchart for the proposed method.

Step 1: A specific number of chromosomes is created as the initial population. In this step, chromosomes with a length equal to the number of tasks (each task represents a gene) are randomly defined which each gene takes a value based on the number of the virtual machine in the cloud provider.

Step 2: Then, the cost associated with each of the chromosomes is determined by the objective functions defined in equations (6) - (8). Since three objective functions are considered at each stage, after each allocation three values are obtained for each chromosome that refers to the cost of the chromosome with respect to the three objective functions.

Step 3: In most cases, there are points that have no superiority over each other and cannot be compared with the concept of domination. Therefore, in order to get the best solution, they should be sorted according to a certain standard. In this algorithm, a rank is assigned to each solution which is based on the number of dominations over the other points. At the end of the algorithm, the points with the best rank (i.e., 1) are selected as the set of solutions or points of the Pareto front.

Step 4: After determining the cost for all chromosomes, some of them are randomly selected for the production of children. Then, for the crossover operator, two parents must be selected at each stage. Here, one-point crossover is used. So that a point is randomly selected in the chromosomes and the two chromosomes are swapped based on that location. Some of the offspring generated after the crossover are selected for the mutation operation. The mutation operation is very useful for escaping from trapping in a local optimum. Of course, it is necessary to select the mutation rate correctly. The mutation operation is performed in a way that first a chromosome is randomly selected for the operation. Then, one of the genes is selected and its value is randomly changed.

Step 5: At this step, the initial population, the population derived from the crossover and the mutation are combined and form a larger 2N population.

Step 6: After combining the populations, chromosomes' fitness is calculated based on the three defined cost functions. At this point, some parents and children should be eliminated so that the number of the main population remains constant. Therefore, first, the total population is sorted according to the rank, and then, according to the crowding distance, a specific number of the individuals with a better situation in terms of the rank and the crowding distance are selected and the rest will be removed so that the number of the main population stays constant and the algorithm continues as before.

Step 7: If the termination criterion is met (the number of iterations), the algorithm ends, otherwise it will enter the next cycle.

IV. EXPERIMENTAL RESULTS

In this section, the proposed method is tested and evaluated. In order to ensure the efficiency of the proposed method, the results are compared with the multi-objective particle swarm optimization algorithm introduced in [27]. The tests were carried out in MATLAB simulation environment on a system with a 2.4 GHz Processor and 4GB RAM.

In order to simulate the proposed algorithm, an environment is designed with 3 cloud providers, 5 virtual machines (VMs), and 10 tasks. It is assumed that each virtual machine belongs to a provider. Also, the maximum capacity of each provider, the cost of a virtual machine unit for each provider, the number of virtual machines allocated to each provider, and the set of virtual machines allocated to each provider varies in each run.

Data and information on virtual machines is presented in Table I. The number of tasks used in the tests is 100, each with a length of {25000, 250000}, a file size of 300, and random output size of 300 which are randomly generated.

TABLE I
VIRTUAL MACHINE PROPERTIES

Number of CPUS	Bandwidth	Memory	Image size	MIPS	VM's Number
4	10000	512	10000	256	1
1	1000	256	1000	300	2
2	10000	512	1000	256	3
1	1000	512	1000	256	4
1	10	256	100	256	5

The parameters of the multi-objective NSGA-II have a direct impact on the algorithm's efficiency and computation time. Through multiple tests, the best value for the parameters of the algorithm was obtained in accordance with Table II. By adjusting the algorithm to the obtained values, the balance between the convergence rate and the minimized cost is established and as a result, the solution is near-optimum in terms of the execution cost and the execution time.

TABLE II
MULTI-OBJECTIVE NSGA II PARAMETERS

Parameter	Maximum iteration	Number of population	Crossover rate	Mutation rate
value	100	20	0.7	0.4

The execution time, the transfer time, and the execution cost plots are shown in Figures 6-8, respectively. The horizontal axis represents iteration and the vertical axis represents the time or the cost. Also, the final values of the cost functions are summarized in Table III for a precise representation of the output of the proposed method.

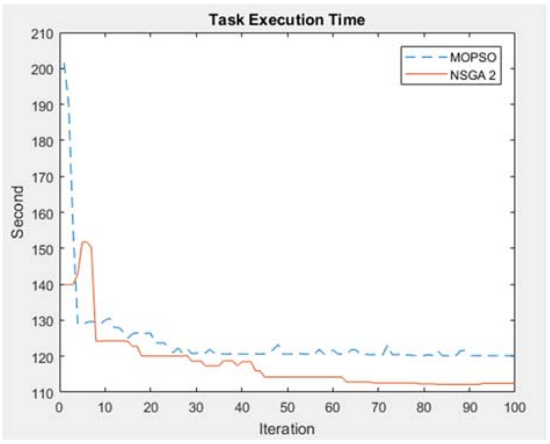

Fig. 6. Total task execution time.

Given the results obtained as well as the statistical values summarized in Table III, we can conclude that the proposed method, i.e. multi-objective NSGA-II is well suited for allocating tasks to virtual machines. So that the minimum values are 112.46 for execution time, 2410 for transfer time and 242.94 for execution cost. Considering the results, it is clear that the proposed method is far better than the Multi-objective particle swarm optimization algorithm with a lower cost at each iteration. Therefore, it can be said that the proposed method has good stability. The uniformity of execution in iterations of the algorithm indicates the stability of the proposed approach.

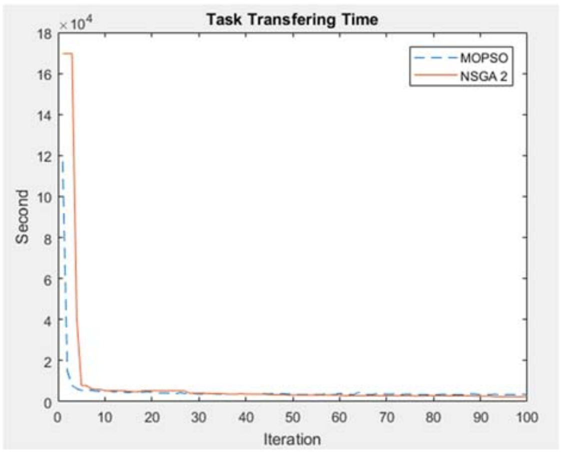

Fig. 7. Total task transfer time.

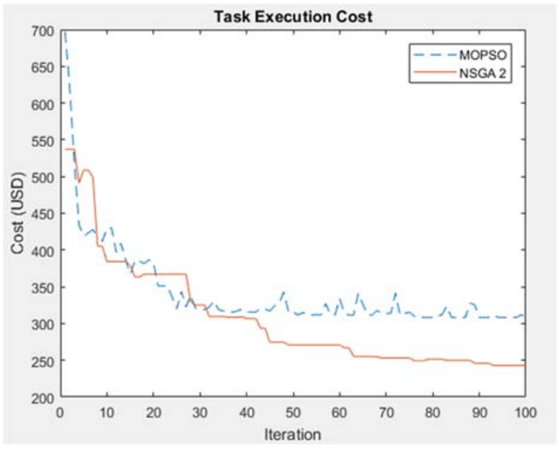

Fig. 8. The total task execution cost.

TABLE III
FINAL VALUES FOR THE OBJECTIVES OF THE FRAMEWORK

	Execution cost	Transfer time	Execution time
MOPSO	308.3	3400	120.17
NSGA II	242.94	2410	112.46

V. CONCLUSION AND FUTURE WORK

In this paper, we presented a multi-objective optimization approach for improving the task scheduling efficiency in the cloud computing. We considered several aspects of the task scheduling problem in the form of objective functions. The state-of-the-art works usually assume the objectives in the task scheduling problem have the same tendency, and therefore, proposed a single-objective solution for the problem. However, we demonstrated the objectives are in conflict with each other, which demand a multi-objective optimization algorithm to establish a trade-off between them. So, we proposed a framework consisted of three objectives called Execution Time, Transfer Time, and Cost, and applied the NSGA-II evolutionary algorithm to find the best scheduling assignment for the given tasks. The obtained results are promising compared to the baseline, an efficient multi-objective task scheduling algorithm based on MOPSO. Our algorithm showed a stable performance with a significant amount of efficiency. In future work, we will study the performance of our approach with respect to scalability.

REFERENCES

1. Rittinghouse, J.W. and Ransome, J.F., 2016. Cloud computing: implementation, management, and security. CRC press.

2. NRaghava, N.S. and Singh, D., 2014. Comparative study on load balancing techniques in cloud computing. Open journal of mobile computing and cloud computing, 1(1), pp. 31-42.

3. Sidhu, A.K. and Kinger, S., 2013. Analysis of load balancing techniques in cloud computing. International Journal of computers & technology, 4(2), pp.737-741.

4. Razaque, A., Vennapusa, N.R., Soni, N. and Janapati, G.S., 2016, April. Task scheduling in cloud computing. In 2016 IEEE Long Island Systems, Applications and Technology Conference (LISAT) (pp. 1-5). IEEE.

5. Agarwal, M. and Srivastava, G.M.S., 2016, April. A genetic algorithm inspired task scheduling in cloud computing. In 2016 International Conference on Computing, Communication and Automation (ICCCA) (pp. 364-367). IEEE.

6. Shojafar, M., Javanmardi, S., Abolfazli, S. and Cordeschi, N., 2015. FUGE: A joint meta-heuristic approach to cloud job scheduling algorithm using fuzzy theory and a genetic method. Cluster Computing, 18(2), pp.829-844.

7. Liu, J., Luo, X.G., Zhang, X.M., Zhang, F. and Li, B.N., 2013. Job scheduling model for cloud computing based on multi-objective genetic algorithm. International Journal of Computer Science Issues (IJCSI), 10(1), p.134.

8. Akilandeswari P. and Srimathi, H., 2016. Dynamic Scheduling in Cloud Computing using Particle Swarm Optimization. Indian Journal of Science and Technology, 9, pp. 120-127.

9. Gupta, P. and Ghrera, S.P., 2016, February. Trust and deadline aware scheduling algorithm for cloud infrastructure using ant colony optimization. In 2016 International Conference on Innovation and Challenges in Cyber Security (ICICCS-INBUSH) (pp. 187-191). IEEE.

10. Masdari, M., ValiKardan, S., Shahi, Z. and Azar, S.I., 2016. Towards workflow scheduling in cloud computing: a comprehensive analysis. Journal of Network and Computer Applications, 66, pp.64-82.

11. Al Nuaimi, K., Mohamed, N., Al Nuaimi, M. and Al-Jaroodi, J., 2012, December. A survey of load balancing in cloud computing: Challenges and algorithms. In 2012 second symposium on network cloud computing and applications (pp. 137-142). IEEE.

12. LD, D.B. and Krishna, P.V., 2013. Honey bee behavior inspired load balancing of tasks in cloud computing environments. Applied soft computing, 13(5), pp.2292-2303.

13. Manakattu, S.S. and Kumar, S.M., 2012, August. An improved biased random sampling algorithm for load balancing in cloud based systems. In Proceedings of the International Conference on Advances in Computing, Communications and Informatics (pp. 459-462).

14. Panwar, R. and Mallick, B., 2015. A comparative study of load balancing algorithms in cloud computing. International Journal of Computer Applications, 117(24).

15. Degtyarev, A. and Gankevich, I., 2016. Balancing load on a multiprocessor system with event-driven approach. In Transactions on Computational Science XXVII (pp. 35-52). Springer, Berlin, Heidelberg.

16. Nakai, A.M., Madeira, E. and Buzato, L.E., 2011, April. Load balancing for internet distributed services using limited redirection rates. In 2011 5th Latin-American Symposium on Dependable Computing (pp. 156-165). IEEE.

17. Sethi, S., Sahu, A. and Jena, S.K., 2012. Efficient load balancing in cloud computing using fuzzy logic. IOSR Journal of Engineering, 2(7), pp.65-71.

18. Ylä-Outinen, P., Latvala, M., Lahtinen, L., Tuunanen, H., Westman, I. and Höneisen, B., Nokia Technologies Oy, 2016. Message-based conveyance of load control information. U.S. Patent 9,369,498.

19. Etminani, K. and Naghibzadeh, M., 2007, September. A min-min max-min selective algorihtm for grid task scheduling. In 2007 3rd IEEE/IFIP international conference in central Asia on internet (pp. 1-7). IEEE.

20. Nace, D. and Pióro, M., 2008. Max-min fairness and its applications to routing and load-balancing in communication networks: a tutorial. IEEE Communications Surveys & Tutorials, 10(4), pp.5-17.

21. Wang, S.C., Yan, K.Q., Liao, W.P. and Wang, S.S., 2010, July. Towards a load balancing in a three-level cloud computing network. In 2010 3rd international conference on computer science and information technology (Vol. 1, pp. 108-113). IEEE.

22. Lu, Y., Xie, Q., Kliot, G., Geller, A., Larus, J.R. and Greenberg, A., 2011. Join-Idle-Queue: A novel load balancing algorithm for dynamically scalable web services. Performance Evaluation, 68(11), pp.1056-1071.

23. Bhatt, H.H. and Bheda, H.A., 2015, September. Enhance load balancing using Flexible load sharing in cloud computing. In 2015 1st International Conference on Next Generation Computing Technologies (NGCT) (pp. 72-76). IEEE.

24. Mohammadi, M. and Rahmani, A.M., 2017. De-centralised dynamic task scheduling using hill climbing algorithm in cloud computing environments.

International Journal of Cloud Computing, 6(1), pp.79-94.

25. Deb, K., Agrawal, S., Pratap, A. and Meyarivan, T., 2000, September. A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. In International conference on parallel problem solving from nature (pp. 849-858). Springer, Berlin, Heidelberg.

26. Deb, K., 2015. Multi-objective evolutionary algorithms. In Springer Handbook of Computational Intelligence (pp. 995-1015). Springer, Berlin, Heidelberg.

27. Alkayal, E.S., Jennings, N.R. and Abulkhair, M.F., 2016, November. Efficient task scheduling multi-objective particle swarm optimization in cloud computing. In 2016 IEEE 41st Conference on Local Computer Networks Workshops (LCN Workshops) (pp. 17-24). IEEE.