A New Task Scheduling Algorithm using Firefly and Simulated Annealing Algorithms in Cloud Computing

Fakhrosadat Fanian
Department of Computer Engineering, Kerman Branch, Islamic Azad University, Kerman, Iran

Vahid Khatibi Bardsiri
Department of Computer Engineering, Bardsir Branch, Islamic Azad University, Kerman, Iran

Mohammad Shokouhifar
Department of Electrical Engineering, Shahid Beheshti University G.C. Tehran, Iran

***Abstract*—Task scheduling is a challenging and important issue, which considering increases in data sizes and large volumes of data, has turned into an NP-hard problem. This has attracted the attention of many researchers throughout the world since cloud environments are in fact homogenous systems for maintaining and processing practical applications needed by users. Thus, task scheduling has become extremely important in order to provide better services to users. In this regard, the present study aims at providing a new task-scheduling algorithm using both firefly and simulated annealing algorithms. This algorithm takes advantage of the merits of both firefly and simulated annealing algorithms. Moreover, efforts have been made in regards to changing the primary population or primary solutions for the firefly algorithm. The presented algorithm uses a better primary solution. Local search was another aspect considered for the new algorithm. The presented algorithm was compared and evaluated against common algorithms. As indicated by the results, compared to other algorithms, the presented method performs effectively better in reducing to make span using different number of tasks and virtual machines.**

Keywords*—Firefly; make span; simulated annealing; task scheduling; cloud

I. Introduction

Cloud computing has recently been introduced as a new technology for users. From a historical perspective, the first computers used were those of the first generation, mainly the mainframes. As time went by, these computers became smaller with higher processing power until personal computers were developed and distributed amongst all users. Next, the technology of networks providing higher processing power emerged by connecting a few small personal computers. However, processing requirements increased exponentially and the need for bigger computing systems became crucially essential.

Thus, smaller networks were privately joined to form bigger networks across the internet. By then, millions of users had access to the internet mostly never using their computers processing power to its full capacity and preferring to give away the idle processing time of their computers to be used for computational tasks. Therefore, many small computational resources were connected; however, it was not possible to completely use these sources within the created network, since these computers were not purposefully created to handle commercial applications. This led to the establishment of a new approach. An approach in which the details were hidden from the user and users did not need to allocate or control infrastructural cloud technologies they were using [1].

In layman's terms, cloud computing was a new user-driven model based on users demands with easy access to flexible and configurable computational sources such as networks, servers, storage areas, practical applications, and services, such that this access is rapidly made with the minimum need for resource management or intervention by the service provider. In general, cloud-computing users are not proprietors of the cloud infrastructure, but rather rent these services from third parties in order to avoid large costs [2]. These users utilize the existing resources in the form of services and only pay for whichever sources they are using [3]. Like any other public service, the costs are based on the amount of service the user requires [4]. Hence, considering that hundreds of people make use of virtual machines, manual allocation of computational sources for different tasks is very troublesome in cloud technology [5]. This highlights the need for an efficient algorithm for task scheduling in cloud environments. This scheduler must be consistent with environmental changes and change in task types [6]. At any moment, millions of users are demanding cloud resources. Scheduling this number of tasks is a serious challenge in cloud processing environments, especially since allocation of optimized resources or task scheduling in clouds must be done in accordance with optimized number and need of systems within the cloud environment so as to maintain the clouds integrity. On the other hand, this scheduling must be done in a way minimizing energy consumption within the cloud. Ergo, this study tries to present an efficient algorithm for task scheduling in clouds using the combination of both firefly and simulated annealing optimization algorithms. This study is organized as follows: Section II reviews related and previous works. Section III discusses and presents a new method. Section IV contains the results of the presented algorithm, and finally Section V gives a conclusion of the entire study.

II. Review of Literature

Cloud computing is currently made up of various aspects, making it a challenging subject. Thus, many researchers have made efforts to investigate the various aspects of cloud computing [7] and have tried to make virtualization and automation technologies focus on improving services in clouds. In this regard, task scheduling and reducing energy consumption in clouds is a very challenging issue for these

environments. Kusic [8] investigated the issue of energy management in virtual heterogeneous environments and used Kalman filters as a method complying with system demands and as a means for prediction and actual implementation.

Kalman filters are used for estimating future demands in order to predict system status and allocate resources accordingly. On the other hand, some researchers focused on the effects of scheduling virtual machines on I/O virtual performance and emphasized on monitoring optimization for better I/O performance. For instance, Ongaro et al. [9] studied the effect of virtual machine observer on performance and presented an idea for arranging processors in an executional queue based on remaining and current value. They ultimately presented an optimization algorithm for scheduling even I/O distribution. However, this scheduling procedure did not take into account the workload and the reallocation of virtual machines. In [10], Kim presented a task-aware scheduler with an emphasis on developing I/O performance.

This scheduler did not consider the heterogeneous workload and variety of weights only focusing on I/O performance. Liao [11] presented a scheduler for scheduling real time applications for supporting respond time, and instead of placing the processor at the end of the executive queue, this method compute the state in which the virtual processor is inserted based on its delay. Goiri [12] presented a task dynamic scheduling policy for allocating informed sources at cloud data centers. The presented scheduler worked to stabilize workload by connecting large tasks of individual devices with necessary hardware, in order to maintain service quality. In other words, these methods reduced energy consumption at data centers turning off servers. Wood [13] presented a virtual machine-driven scheduling policy based on using resources including processor, memory, and subnet components. However, instead of optimizing and scheduling operational energy, his study mainly focused on developing an algorithm for avoiding local traps. Dorigo et al. presented the ACO algorithm [14]. The ACO was a random search algorithm, which used positive feedback and followed actual ant colony behavior. In [15] this algorithm was used to allocate optimized sources for tasks in a dynamic cloud environment in order to minimize make span.

Liu et al. [16] worked on a scheduling algorithm based on genetic and ant colony algorithms. They tried to make use of the advantages of both algorithms. This algorithm uses the global search in genetic algorithm in order to reach the optimized solution faster. It also utilizes initial values for pheromones in the ACO algorithm. Guo et al. [17] used a formulated particle swarm optimization (PSO) model for minimizing process costs. They also tried to use crossover and mutation functions of the genetic algorithm along with the PSO model. Lakro et al. [18] investigated various variables and their optimization in cloud computing environments. They tried to present a multi-variable optimization algorithm for scheduling and improving performance of data centers. Jia et al. [19] investigated scheduling of various tasks of different sizes on a set of parallel batch machine and presented a meta-heuristic algorithm based on max-min and ant system for minimizing make span.

III. Methodology and Suggested Algorithm

Cloud computing is one of the newest technologies today, which allows users to send their requests to clouds and pay a certain amount of fees based on the service provided. On the other hand, cloud environments are in fact homogenous systems suitably storing large applications and data for services. Considering this, scheduling of these data and large applications in these systems is of great importance. The present study tries to present a new algorithm based on firefly and simulated annealing algorithms called FA-SA in order to schedule tasks in clouds. The details of the suggested combination are expressed below. The general framework for this study is shown in Fig. 1.

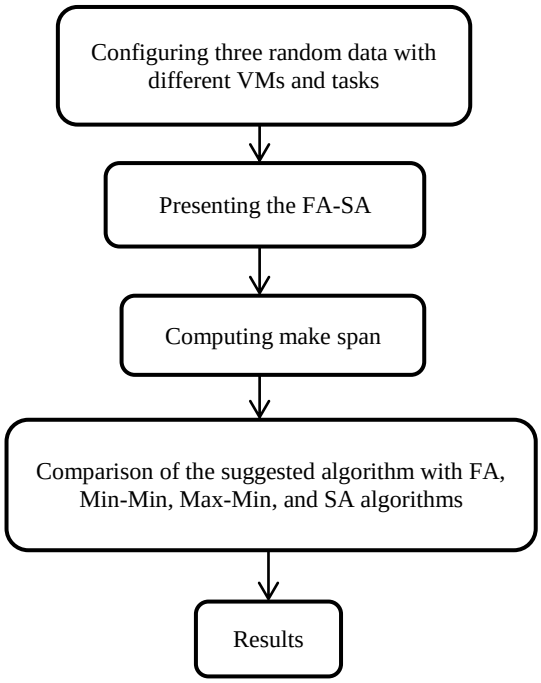

Fig. 1. General framework for the study.

A. Problem Statement

The allocation of tasks to virtual machines in cloud computing systems is a problem, in which m number of tasks, $V= \{t_1, t_2, \ldots, t_m\}$ are to be allocated to certain virtual machines. In this study, the total number of tasks are randomly selected from 10 to 100 tasks and categorized into three different data sets with different number of virtual machines. The tasks are made randomly. Also $P= \{ v_1, v_2, \ldots , v_n \}$ are the n virtual machines used. All systems are the same, meaning tasks are performed in a homogeneous environment.

B. Possible Solutions

This study uses a combination of firefly algorithm (FA) and simulated annealing (SA). The feasible solution in this study is a string of m characters, where m is the total number of tasks. According to (1), if task i is allocated to a virtual machine, j, the i_{th} place in the relative string, has a value of "j". 20 virtual machines are considered for all m tasks. A feasible solution for the problem is shown in Fig. 2.

$$Solution_i = j \qquad \text{if task } i \text{ is allocated to machine } j \qquad (1)$$

	1	2	3	4	5	6	7	....	M
A Solution:	13	1	17	8	2	6	14	....	7

Fig. 2. A feasible solution: for example, the first task is given to machine I3 and the second task is given to machine 1.

C. Objective function in the Suggested Algorithms (FA, SA and FA-SA)

As previously mentioned, an objective function is needed for all algorithms in order to schedule tasks and minimize amount of make span. Task scheduling is an optimization problem in which tasks are to be allocated to sources at certain times. In other words: *n* tasks, j_1, j_2, …, j_n, with different sizes are to be allocated to *m* identical scheduler machines such that make span is minimized. Make span is defined as the total amount of time required to perform all tasks (after all tasks have been done). Recently, this problem has been introduced as a dynamic scheduling problem, in which for every task, the dynamic algorithm must use the existing information to make a decision before the next task comes. This is one of the most famous dynamic problems and the first for which a competitive analysis was presented by Graham in 1966 [20].

D. Overall Stages of Allocating Tasks to Virtual Machines using the Suggested FA-SA Algorithm

Evolutionary algorithms are generally based on population and make use of a very suitable global search strategy. The firefly algorithm [21] was used in this study. This algorithm is a meta-heuristic algorithm inspired by the behavior and motion of fireflies in nature. This algorithm is similar to other population-based algorithms and computes the optimized solution (or near to optimized) in an iterative manner. The algorithm starts by performing a search procedure in a randomly developed population. Each member of the population (location of each firefly in the search space) is a possible solution for the problem, which is shown in Fig. 2 according to (1). Each iteration in the FA algorithm has two main stages: Stage 1, evaluating the suitability of the solutions and Stage 2, updating the population (establishing a new population). These two stages are continuously performed in iteration until the termination criteria of the algorithm is satisfied.

The termination condition in this study is the completion of all tasks. The FA algorithm is a population-based algorithm with the ability to perform a very suitable global search since it has a very high convergence rate and each firefly tries to find the best state individually; thus, it avoids local optimums and searches for the global optimum [22]. On the other hand, the SA algorithm has a very convenient local search procedure. It is for this reason that both of these algorithms were combined in this study to form the FA-SA algorithm in order to benefits from the advantages of both of these algorithms for performing a better scheduling of tasks in clouds.

In the presented method, the FA algorithm initiates first in order to perform a global search in the search space. After the FA algorithm, the SA algorithm is executed to perform a local search near the previous solution provided by the FA algorithm. In other words, the initial population for the SA algorithm is not selected randomly, rather it gets the value provided by the FA algorithm which is in fact the optimum value provided by the FA algorithm. The general flowchart for the suggested method is shown in Fig. 3. The stages of the suggested algorithm will be explained in more detail in the following section.

a) Producing a random initial population for the FA algorithm): As previously mentioned, the first stage for all evolutionary algorithms is producing initial solutions, which are mostly done randomly. The initial solutions for the FA algorithm in this study are produced considering the following regulations:

1) Perform the following stages for m iterations (where m is the total number of tasks): find the virtual machine(s) with the least termination time (since the data are random multiple machines may have the same value).

2) Perform the following stages for m iterations (where m is the total number of tasks): find the virtual machine(s) with the least termination time (since the data are random multiple machines may have the same value).

3) If a virtual machine is found, select the virtual machine, otherwise randomly select a virtual machine with the least termination time (since data are random multiple machines may produce the same value).

4) Search the initial data set (containing tasks and virtual machines) and find the virtual machine selected in stage 2 and choose the task with the least time from the unallocated tasks for that machine.

5) If a task exits with the least amount of time, select that task; otherwise, randomly select a task.

6) All tasks are assigned?

7) no, go to stage 1, otherwise terminate.

In other words, each task is allocated to a virtual machine according to the regulations mentioned above. It is worth noting that since the data sets of this study are random in nature and according to the regulations, random selection is performed two times, the initial population or rather the initial solutions are different for each iteration, though due to the nature of the regulations, these initial solutions are near optimum.

b) Competency assessment for produced solutions: The solutions produced by the FA algorithm are evaluated in each iteration after the population has been updated. This evaluation works on the basis of the objective function. In order to evaluate each member of the population (each firefly), allocated tasks for each machine are considered first. Next, execution time on each machine is computed and finally termination time for all tasks are computed.

c) Updating population in the FA Algorithm: The firefly algorithm was presented by Yang [23] and is inspired by the motion and behavior of fireflies in nature. Fireflies produce short and rhythmic lights. These rhythmic lights, light radiation

rate, and distance are what make two fireflies attract each other. Light intensity at a distance of r from the light source has a relationship with the reverse squared amount of distance. In the firefly algorithm, light can be considered as the objective function to be optimized. In short, the firefly algorithm is based on the following three principles:

1) All fireflies are unisexual and each firefly attracts the other firefly despite their sexuality.

2) Attraction of fireflies is proportional to their radiance such that the firefly with less light intensity is attracted to the one with higher light intensity, and if there is no firefly with a higher light intensity in the locality, fireflies move randomly. 3- The light intensity of fireflies is determined as the objective function [24], [25]. In FA algorithm, the location of each firefly in m-dimension space determines a solution for the optimization problem, where m is the number of optimization variables (total number of virtual machines). Considering that fireflies' location is defined in a continuous space, this study considers the location of each firefly within the (0,n] range, where n is the total number of machines. Therefore, each dimension value for each firefly is a value from 0 to n. In each iteration of the evaluation stage, each dimension for each firefly is rounded up to the nearest natural number that is bigger than the current number.

Therefore, evaluation of fireflies takes place in a discrete space. However, fireflies' motion and attraction are done continuously. After determining the time for the solution of each firefly using relative objective function, radiance of each firefly i is computed using (2) (since radiance in this algorithm denotes higher competency, every firefly with a lower objective function has a higher f_i), where $objective\ function_{\ i}$ and f_i denote error rate (objective function) and radiance for the ith firefly, respectively. Each iteration selects fireflies with the highest radiance. Then, each of the remaining fireflies moves towards the nearest radiant firefly. The distance between firefly i and firefly j is computed by (3):

$$f_i = \frac{1}{\text{objective function}_{\ i}} \quad (2)$$

$$r_{ij} = \|x_i - x_j\| = \sqrt{\sum_{k=1}^{d} (x_{i,k} - x_{j,k})^2} \quad (3)$$

where x_i and x_j are the locations of the i_{th} and j_{th} fireflies, respectively. d is the number of optimization variables, which in this case is equal to the total number of tasks. Movement of firefly i towards firefly j is formulated as (4). The second expression in this statement shows the attraction if firefly i towards firefly j and the third expression shows a random movement in the attraction procedure. α and β are two static variables that configure the effect of the two expressions when firefly i moves. η determines the way fireflies move and is usually selected between 0 and infinity.

$$x_i = x_i + \beta * e^{-\eta r_{ij}}(x_j - x_i) + \alpha * (rand - 0.5) \quad (4)$$

d) Addition of local search to the FA algorithm: Three types of local searches were added to the firefly algorithm in this study, where each type is used with a probability of 1/3 for each iteration (generation). These searches include exchange mutation, inverted exchange mutation, and a suggested local search called hybrid max-min to exchange (HHME). These procedures are explained in more detail in the following section.

- Exchange mutation: In this procedure, two machines are randomly selected and their tasks are exchanged [26]. Fig. 4 shows the search procedure used in the proposed algorithm.

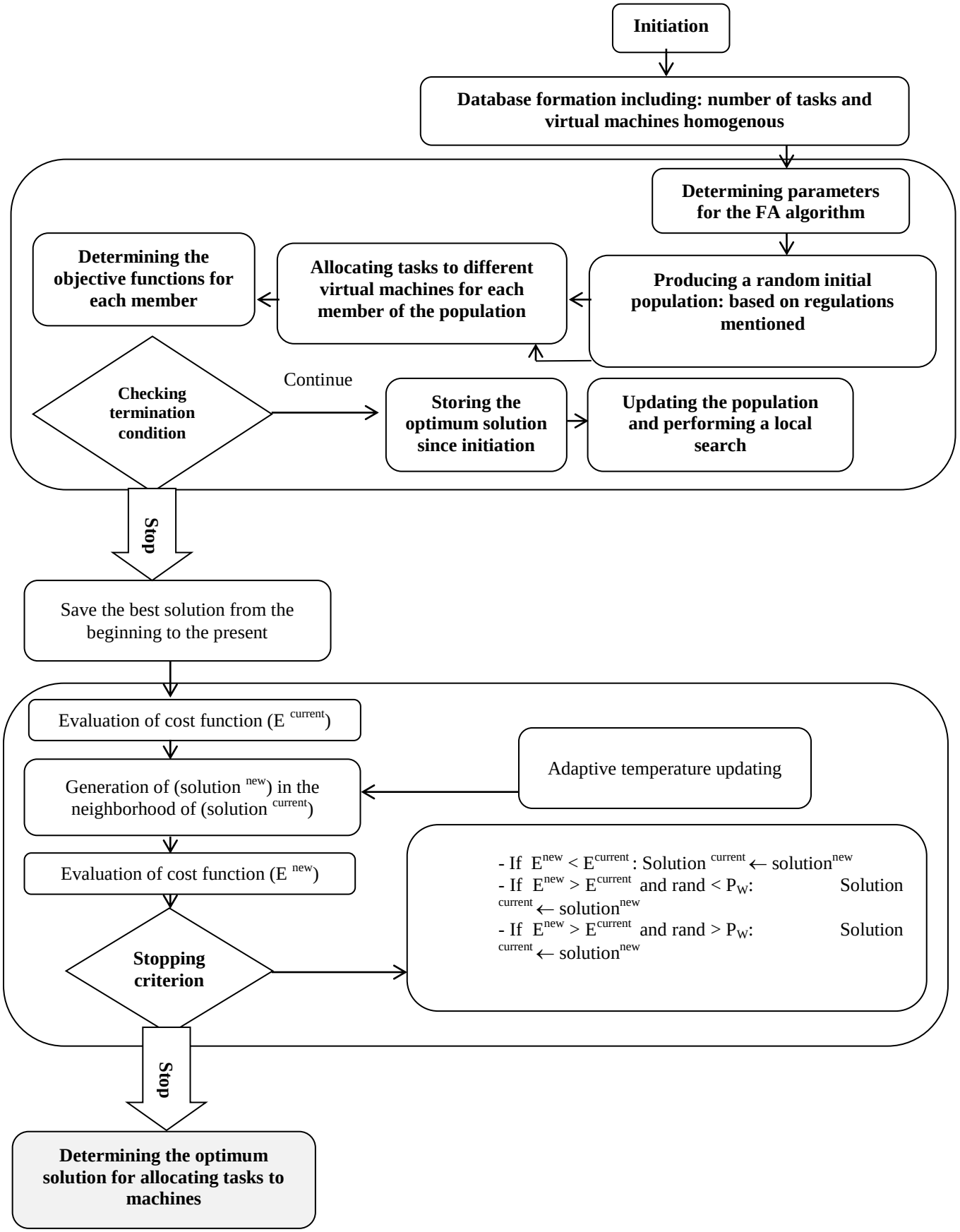

Fig. 3. General flowchart for the FA-SA algorithm.

Fig. 4. Search procedure used in the suggested algorithm, before exchange mutation (current solution) and after exchange mutation (new solution).

- Inverted exchange mutation: two machines are randomly selected and their tasks are inverted [26]. Fig. 5. shows the search procedure used in the proposed algorithm, before inverted exchange mutation (current solution) and after the inverted exchange mutation (new solution).

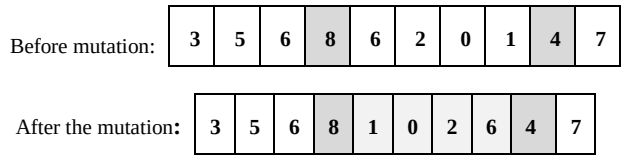

Fig. 5. search procedure used in the suggested algorithm, before inverted exchange mutation (current solution) and after inverted exchange mutation (new solution).

- Hybrid max-min to exchange (HMME): In this procedure, virtual machines with the highest and lowest value of termination time are selected and; 1) a minimum task from the machine with the highest termination time is transferred to the machine with the lowest termination time or 2) a task is randomly selected from the machine with the highest termination time and transferred to the machine with the lowest termination time or 3) a maximum task from the machine with the highest termination time is transferred to the machine with the lowest termination time.

e) Local search using SA algorithm: the simulated annealing, presented in 1983 [27] algorithm, is an optimization algorithm that uses local search. The gradual annealing technique is used by metallurgists in order to reach a state where the solid material is sorted properly with minimized energy. In this technique, the substance is placed at high temperature then cooled down gradually. During this algorithm, each state s in the search space is similar to a state of a physical system and the E(s) function which must be minimized is similar to the internal energy of the system in that specific state. The purpose of this procedure is to transfer the system from its initial random state to a state where the system has the lowest amount of energy.

For an optimization problem, the algorithm starts with a random initial solution and gradually moves towards neighboring solutions in an iterative manner. In each iteration, if the neighbor solution ($solution^{new}$) is better than the current solution ($solution^{current}$), the algorithm selects the former solution as the new current solution. Otherwise, the algorithm selects the new solution with a probability of of $p_w = exp(-\Delta E/T)$, where $\Delta E = E^{new} - E^{current}$ is the difference between the objective function value of the current solution and that of the neighboring solution and T is the temperature variable. This algorithm iterates for each temperature, and gradually decreases the temperature. The temperature is initially high so that the possibility of choosing worse solutions is high. However, with the gradual decrease in temperature, the possibility of choosing worse solutions decreases and better solutions are selected. Therefore, the algorithm converges to a proper solution. As seen in Fig. 6, in this study, random changes in one dimension of the solution have been selected for local search. The possibility for performing this procedure is defined as P_m, where the value of each dimension in the current solution changes with the probability of P_m.

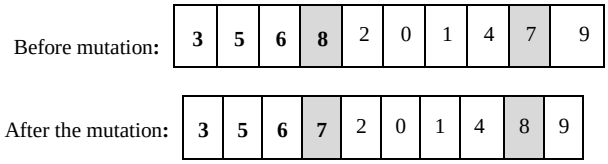

Fig. 6. Local search in the SA algorithm, before exchange mutation (current solution) and after exchange mutation (new solution).

IV. Results

The suggested FA-SA algorithm was simulated in MATLAB and was compared with three data sets and the following algorithms: min-min, max-min, firefly, and simulated annealing. Comparison results are provided below.

A. Datasets

Three different datasets were randomly selected in this study considering that the minimum and maximum numbers of tasks were 10 and 100, respectively. The number of tasks and machines were chosen randomly such that the first dataset, name data1, contained 100 tasks and 8 homogeneous virtual machines randomly selected according to the mentioned criteria. Detailed specification of these three datasets are provided in Table I.

TABLE I. Specifications of Randomly Selected Datasets

Type data	Number of task	Number of VM	Min_task	Max_task
Data1	100	8	10	100
Data2	200	20	10	100
Data3	500	20	10	100

B. Parameter Configuration for Optimization Algorithms

Configuration circumstances for the FA, SA, and FA-SA optimization algorithms are shown in Table II. As can be seen, considering that SA is a single-population algorithm, the number of iterations in SA is more than that of FA and the circumstances for the FA-SA algorithm are a combination of those related to FA and SA, since the final solution of FA algorithm is used as the initial solution for the SA algorithm.

C. Evaluation using Makespan

This section compares the FA-SA algorithm with SA, FA, min-min, and max-min algorithms based on objective function for computing and minimizing make span. Fig. 7 shows the evaluation results of FA-SA and other algorithms on the data1 dataset. As it is observed, since workload and number of virtual machines is lower compared to other datasets, all algorithms, except max-min, showed a similar make span value and the suggested algorithm outperformed other algorithms in reducing make span. It is worth mentioning that all results were

based on 10 iterations of the algorithms and were expressed as mean values.

TABLE II. CONFIGURABLE PARAMETERS FOR FA AND SA ALGORITHMS

Algorithm (Value)			
Parameter	**FA**	**SA**	**FA-SA**
FA	200		200
FA $_{population}$	50		50
FA $_{alpha}$	0.05		0.05
FA $_{beta}$	2		2
FA $_{gama}$	0.001		0.001
SA $_{Max\_iter}$		500	500
SA $_{T\_initial}$		0.001	0.001
SA $_{T\_final}$		0	0

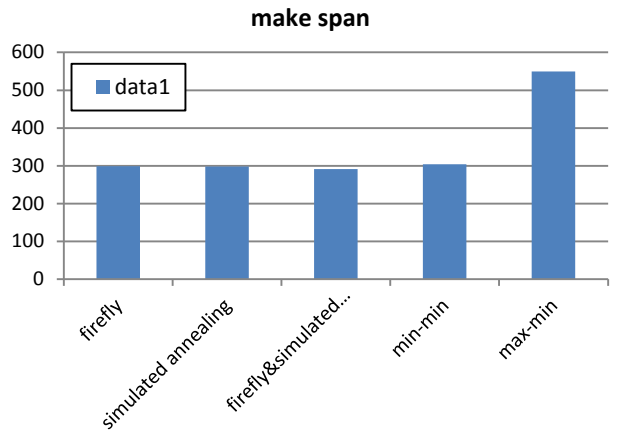

Fig. 7. Comparison of scheduling algorithms on the data1 dataset.

Results of computing make span using the suggested algorithm on the data2 dataset indicate that the FA-SA algorithm was more successful in minimizing make span compared to other algorithms; thus, it can be said that the FA-SA algorithm also creates a good workload balance on virtual machines. Complete results of this comparison are shown in Fig. 8.

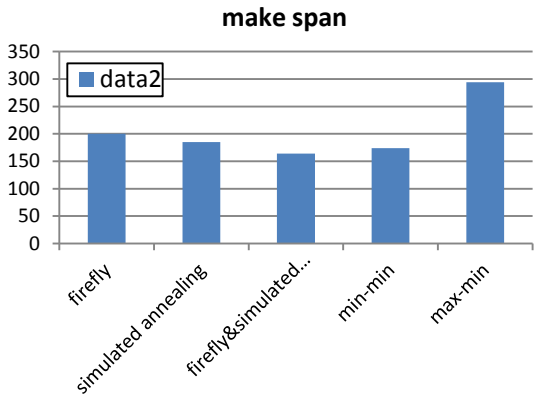

Fig. 8. Comparison of different scheduling algorithms on the data2 dataset.

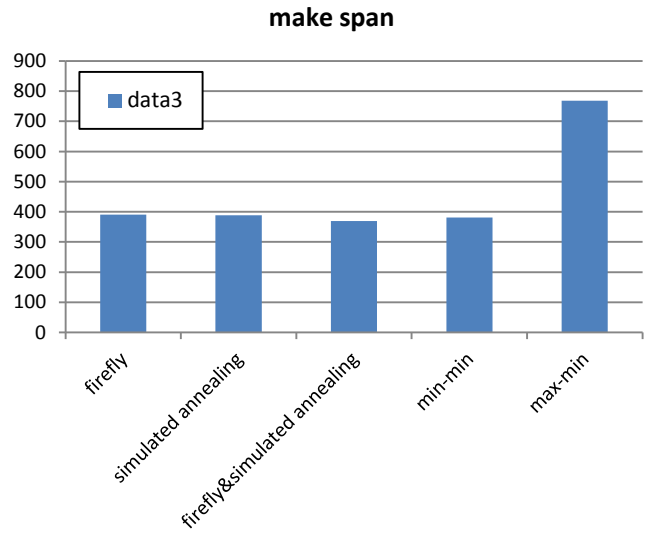

Fig. 9. Comparison of different scheduling algorithms on the data3 dataset.

TABLE III. MAKE SPAN RESULTS FOR EACH OPTIMIZATION ALGORITHM ON ALL THREE DATASETS

Type algorithm	**Data1**	**Data2**	**Data3**
Firefly	299	200	391
Simulated annealing	298	185	388
Firefly& Simulated annealing	291	164	369
Min-min	304	174	381
Max-min	550	294	768

The FA-SA algorithm was further tested on the data3 dataset compared to the previous datasets, had a higher workload. Results of this performance are shown in Fig. 9. These results indicate once again that the FA-SA algorithm is superior to FA, SA, min-min, and max-min algorithms in reducing make span and balancing workload on machines. Overall results of 10 iterations of the algorithms with mean values are shown in Table III.

V. CONCLUSION

Cloud processing, parallel computing and development of distributed computations are all new concepts in computer sciences. One of the major issues in this regard known as a major challenge and an NP-hard problem is the scheduling of tasks in cloud computing. Task scheduling in cloud computing have been discussed in regards to meta-heuristic algorithms such as genetic, ant colony, and other algorithms. However, this study aimed to combine two optimization algorithms, namely the firefly and the simulated annealing algorithms in order to create the new hybrid FA-SA algorithm. Also, a new mechanism for producing initial population and a new method for local search were presented. The suggested algorithm was compared with firefly, simulated annealing, min-min, and max-min algorithms. Results indicated that the FA-SA algorithm can perform much better in reducing make span in different scenarios with different numbers of tasks and virtual machines.

For future works, we will try to focus our attention on energy performance and resource allocation in these systems.

REFERENCES

[1] M. Miller, "Cloud computing: web based applications that change the way you work and collaborate online", Que Publishing, 2008.

[2] Y. Gao, H. Guan and Z. Qi, et al., "A multi-objective ant colony system algorithm for virtual machine placement in cloud computing", Journal of Computer and System Sciences, Vol. 79, pp. 1230 –1242, 2013.

[3] K. Danielson," Distinguishing cloud computing from utility computing", (http:/ / www. ebizq. net/ blogs/ saasweek/ 2008/ 03/ distinguishing_cloud_computing/), 2008.

[4] J. Beliga, R. W. A. Ayre and K. Hinton, et al. "Green cloud computing: balancing energy in processing, storage and transport", Proceedings of the IEEE, Vol. 99, pp.149-167, 2011.

[5] H. Qiyi and H. Tinglei, "An optimistic job scheduling strategy based on QoS for cloud computing", Proceedings of the International Conference on Intelligent Computing and Integrated Systems (ICISS), pp.673-675, 2010.

[6] F. Chang, J. Ren and R. Viswanathan, "Optimal resource allocation for batch testing" ", Proceedings of the International Conference on Software Testing Verification and Validation (ICST), pp.91-100, 2009.

[7] G. Lin, G. Dasmalchi and J. Zhu, "Cloud computing and IT as a service: opportunities and challenges", Proceedings of the International Conference on Web Services (ICWS), pp.1-5, 2008.

[8] D. Kusic, J. O. Kephart and J. E. Hanson, et al. "Power and performance management of virtualized computing environments via lookahead control", Cluster Computing, Vol. 12, pp. 1-15, 2009.

[9] D. Ongaro, A. L. Cox and S. Rixner, "Scheduling I/O in virtual machine monitors", Proceedings of the Fourth ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments, pp.1-10, 2008.

[10] H. Kim, H. Lim and J. Jeong, et al. "Task-aware virtual machine scheduling for I/O performance", Proceedings of the ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments, pp. 101-110, 2009.

[11] G. Liao, D. Guo and L. Bhuyan, et al. "Software techniques to improve virtualized I/O performance on multi-core systems", Proceedings of the 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS), pp. 161-170, 2008.

[12] I. Goiri, F. Julia and R. Nou, et al. "Energy-aware scheduling in virtualized datacenters", IEEE International Conference on Cluster Computing (cluster), pp. 58 – 67, 2010.

[13] T. Wood, P. Shenoy and A. Venkataramani, et al. "Black-box and gray-box strategies for virtual machine migration", Proceedings of the 4th USENIX conference on Networked systems design & implementation (NSDl), pp.17-17, 2007.

[14] M. Dorigo and C. Blum, "Ant colony optimization theory: A survey", Theoretical Computer Science Vol.344, pp.243–278, 2005.

[15] M. A. Tawfeek, A.El-Sisi and A. E. keshk, et al.," Cloud task scheduling based on ant colony optimization", Proceedings of the 8th International Conference on Computer Engineering & Systems (ICCES), pp. 64 – 69, 2013.

[16] C.Y. Liu, C. M. Zou and P. Wu, "A task scheduling algorithm based on genetic algorithm and ant colony optimization in cloud computing", Distributed Computing and Applications to Business, Engineering and Science (DCABES), pp. 68-72, 2014.

[17] L. Guo, S. Zhao, and S. Shenet, et al. " Task Scheduling Optimization in Cloud Computing Based on Heuristic Algorithm", Journal of networks, Vol. 7, pp.547-553, 2012.

[18] A. V. Lakra and D. K. Yadav, "Multi-objective tasks scheduling algorithm for cloud computing throughput optimization", Proceedings of the International Intelligent Computing, Communication & Convergence (ICCC), pp. 107 – 113, 2015.

[19] Z. H. Jia, C. Wang and J. Y. T. Leung," An ACO algorithm for makespan minimization in parallel batch machines with non-identical job sizes and incompatible job families", Applied Soft Computing, vol.38, pp.395-404, 2016.

[20] R. L. Graham, "Bounds for certain multiprocessing anomalies", Bell System Technical Journal, Vol.45, pp.1563–1581, 1966.

[21] X. S. Yang, "Firefly algorithms for multimodal optimization". Stochastic Algorithms: Foundations and Applications, Vol. 5792, pp. 169–178, 2009.

[22] O. Jafarzadeh-Shirazi, "Task scheduling with firefly algorithm in cloud computing", Science International, Vol.1, pp-167-171, 2014.

[23] S. Yang, "Nature-Inspired Metaheuristic Algorithms", Luniver Press,2010.

[24] X.S. Yang, "Firefly algorithm, stochastic test functions and design optimization," International Journal of Bio Inspired Computation, Vol. 2, pp. 78–84, 2010.

[25] X.S. Yang, "Firefly algorithm, levy flights and global optimization," Research and Development in Intelligent Systems, pp. 209–218, 2010.

[26] K. Deep, H. Mebrahtu, "Combined mutation operators of genetic algorithm for the travelling salesman problem", International Journal of Combinatorial Optimization Problems and Informatics, Vol. 2, pp. 1-23, 2011.

[27] S. Kirkpatrick, C. D. Gelatt and M. P. Vecchi,"Optimization by Simulated Annealing", Science, Vol.220, pp. 671–680, 1983.