A Novel Bee Colony Approach to Distributed Systems Scheduling

Raheleh Sarvizadeh
Young Researchers Club, Islamshahr Branch, Islamic Azad University, Tehran, Iran

Mostafa Haghi Kashani
Department of Computer Engineering, Shahr-e-Qods Branch, Islamic Azad University, Tehran, Iran

Fahimeh Sadat Zakeri
Department of Electrical and Computer Engineering, University of Tehran, Tehran, Iran

Seyed Mahdi Jameii
Department of Computer Engineering, Shahr-e-Qods Branch, Islamic Azad University, Tehran, Iran

ABSTRACT
The problem of task scheduling in distributed systems is known as an NP-hard problem, and methods based on heuristic or metaheuristic search have been proposed to obtain optimal and suboptimal solutions. The scheduling problem is a key factor for distributed systems to gain better performance. In this paper, an efficient method based on memetic algorithm is developed to solve the problem of distributed systems scheduling. With regard to load balancing efficiently, Bee Colony Optimization (BCO) has been applied as local search in the proposed memetic algorithm. The proposed method has been compared to existing GA-based method and two memetic-Based methods in which Tabu method and Learning Automata method have been used as local search. The results demonstrated that the proposed method outperform the above mentioned methods in terms of CPU Utilization, communication cost and Makespan.

General Terms
Distributed Systems, Evolutionary Algorithms.

Keywords
scheduling, memetic algorithm, Bee Colony Optimization.

1. INTRODUCTION
Distributed system scheduling has been a source of challenging problems for researchers in the area of computer engineering. Task scheduling in a distributed system can be defined as allocating tasks to processors of each computer in a way that the optimum performance is obtained. The aim of task scheduling is minimizing Communication Cost and Makespan (task completion time) and also maximizing CPU utilization.

In order to solve scheduling problem, several methods have been proposed. The proposed methods can be generally classified into three categories: Graph-theory-based approaches [1], mathematical models-based methods [2] and heuristic Techniques [3-7]. Task scheduling in distributed systems is known as NP-hard [8]. Therefore using heuristic Techniques can solve this problem more efficiently. Three most well-known heuristics are the iterative improvement algorithms [9], the probabilistic optimization algorithms, and the constructive heuristics. In the probabilistic optimization group, GA-based methods [10-15] are considerable. The main distinction among them is chromosomal representation used for a schedule. However, these approaches scan the entire solution space without considering the techniques that can reduce the complexity of the optimization. In other words, their main shortcoming is spending much time doing scheduling. This shortcoming of GA-based methods can be reduced by combing GA with another optimization technique. Hence, this paper proposes a new algorithm by using memetic algorithm to cope with this shortcoming. Bee Colony Optimization has been applied as local search in the proposed memetic algorithm. The results demonstrated that the proposed method outperform three existifng methods in term of CPU Utilization, communication cost and Makespan.

There are two categories for task scheduling: static and dynamic. In dynamic scheduling, schedules are created during run time without having any knowledge of the task in hand before arriving. While in static scheduling, schedules are created before run time and do not change. Similarly, tasks must be all known in advance. In other words, a static task scheduling algorithm schedules a set of tasks with known processing and communication characteristics on processors to optimize Makespan, Communication Cost, and CPU utilization. In this paper, the author focused on static scheduling.

One of the crucial aspects of the scheduling problem is load balancing. When newly created processes enter the system randomly, some processors may be overloaded heavily while the others are under-loaded or idle. The main objectives of load balancing are to spread load on processors equally, maximizing processors utilization, and minimizing total execution time. In dynamic load balancing, processes must be dynamically allocated to processors in arrival time and obtain a near optimal schedule; therefore, the execution of the dynamic load balancing algorithm should not take too long to arrive at a rapid task assignments decision [8, 16, 17]. In this paper scheduling algorithm considering load balancing has been proposed.

The rest of this paper is organized as follow: In section 2 used method as local search in the proposed memetic algorithm is presented. Proposed method comes in section 3. Experimental results are given in section 4. Section 5 concludes the paper.

2. USED METHOD AS LOCAL SEARCH IN THE PROPOSED ALGORITHM

In this section the method which has been used as local search in the proposed memetic algorithm has been described.

Bee Colony Optimization (BCO) is a metaheuristic for solving combinatorial optimization problems. The BCO is inspired by bees' behavior in the nature. In a honey bee colony forager bees search the environment for flower paths and if they find a good source of food, share it with other bees. While the forager bees come back to the hive they share the found information about food sources by a special movement named waggle dance. The studies on this type of bee dance shows that in the midst of this dance some information like direction, distance, quantity and quality of the food source are shared with respect to other bees [18-20].

The BCO is a population based algorithm. Population of the artificial bees searches for the optimal solution. Every artificial bee generates one solution to the problem. The algorithm consists of two alternating phases: *forward pass* and *backward pass*. During each forward pass, every bee is exploring the search space. It applies a predefined number of moves, which construct and improve the solution, yielding to a new solution.

Having obtained new partial solutions, the bees return to the nest and start the second phase, the so-called backward pass. During the backward pass, all bees share information about their solutions. In nature, bees would perform a dancing ritual, which would inform other bees about the amount of food they have found, and the proximity of the patch to the nest. In the search algorithm, the bees announce the quality of the solution, i.e. the value of objective function. During the backward pass, every bee decides with a certain probability whether it will advertise its solution or not. The bees with better solutions have more chances to advertise their solutions. The remaining bees have to decide whether to continue to explore their own solution in the next forward pass, or to start exploring the neighborhood of one of the solutions being advertised. Similarly, this decision is taken with a probability, so that better solutions have higher probability of being chosen for exploration.

The two phases of the search algorithm, forward and backward pass, are performed iteratively, until a stopping condition is met. The BCO algorithm parameters whose values need to be set prior the algorithm execution are as follows[18]:

B - The number of bees in the hive.

N -The number of constructive moves during one forward pass.

At the beginning of search process, all bees are in the hive. To continue, it is the pseudo code of the BCO algorithm:

1. Initialization: every bee is set to an empty solution;
2. For every bee do the forward pass:
   a) Set k = 1; //counter for constructive moves in the forward pass.
   b) Evaluate all possible constructive moves;
   c) According to evaluation, choose one move using the roulette wheel;
   d) $k = k + 1$; If $k \leq NC$ Go To step b.
3. All bees are back to the hive; // backward pass starts;
4. Sort the bees by their objective function value;
5. Every bee decides randomly whether to continue its own exploration and become a recruiter, or to become a follower (bees with higher objective function value have greater chance to continue its own exploration);
6. For every follower, choose a new solution from recruiters by the roulette wheel;
7. If the stopping condition is not met Go To step 2;
8. Output the best result.

Initial bee colony is constructed by the following algorithm:

1. At each step, a next task to be assigned is selected.
2. The processor (the selected task is going to be assigned to) is determined.
3. Repeat these two steps until all tasks have been assigned to a processor.

The choice is probabilistic bias to a probability function. This function is updated at each iteration in a reinforcement way by using the features of good solutions.

In the relation (1), after selecting a task, a randomly chosen processor will be selected.

$$pro_{ij} = \frac{a_{ij} / Load(p_j)}{\sum_{k=1}^{m} [a_{ik} / Load(p_k)]} \qquad (1)$$

In the above a_{ij} , $1 \leq i \leq n$, $1 \leq j \leq m$ is the execution time of task t_i on processor P_j. The processor with a lower value of the running times has greater probability to be chosen.

All bees return to the hive after generating the partial solutions. All these solutions are then evaluated by all bees. (The latest time point of finishing the last task at any processor characterizes every generated partial solution).

Let us denote by $Load(P_b)$(b=1, 2, ..., B) the latest time point of finishing the last task at any processor in the partial solution generated by the *b*-th bee. We denote by O_b normalized value of the time point $Load(P_b)$, i.e.:

$$O_b = \frac{Makespan - Load(p_b)}{Makespan - Minspan} \qquad (2)$$

In the above *Makespan* and *Minspan* are respectively the smallest and the largest time point among all time points produced by all bees. The probability that *b*-th bee (at the beginning of the new forward pass) is loyal to its previously generated partial solution is expressed as follows:

$$p_b^{u+1} = e^{-\frac{Makespan - O_b}{u}}, \quad b = 1, 2, ..., B \qquad (3)$$

Which *u is* the ordinary number of the forward pass (e.g., *u* = 1 for first forward pass, *u* = 2 for second forward pass, etc.).

For each uncommitted bee with a certain probability it is decided which recruiter it would follow. The probability that *b*'s partial solution would be chosen by any uncommitted bee is equal to:

$$pro_b = \frac{O_b}{\sum_{k=1}^{R} O_k} \quad , b = 1,2,...,R \tag{4}$$

Where O_k represents normalized value for the objective function of the *k*-th advertised partial solution and *R* denotes the number of recruiters. Using relation (4) and a random number generator, each uncommitted follower joins one recruiter.

3. PROPOSED MEMETIC ALGORITHM

In our model there are finite numbers of tasks, each having a task number and an execution time. All tasks are placed in a task pool from which tasks are assigned to processors. Figure 1 shows the proposed chromosome. In this chromosome, tasks 1,2,3,4,5,6,7 and 8 are assigned to processors 2,1,2,3,3,4,2 and 1 respectively. Both tasks 1 and 3 are assigned to processor 2 but, first task 1 is executed then task 3.

1	2	3	4	5	6	7	8
2	1	2	3	3	4	2	1

Fig 1: An example of proposed chromosome

Before describing the proposed methods, it is necessary to state some definitions which have been stated in [21] as follow:

- $T = \{t_1, t_2, t_3, \ldots, t_n\}$ is a set of tasks to execute.
- $P = \{p_1, p_2, p_3, \ldots, p_m\}$ is a set of processors in the distributed system. Each processor can only execute one task at a time, in other words, a processor completes current task before executing a new one, and a task cannot be moved to another processor during execution.
- R is an $m \times m$ matrix, the element r_{uv} $1 \le u, v \le m$ of R, which is the communication delay rate between P_u and P_v.
- H is an $m \times m$ matrix, the element h_{uv} $1 \le u, v \le m$ of H, which is the time required to transmit a unit of data from P_u to P_v. It is obvious that $h_{uu} = 0$ and $r_{uu} = 0$.
- A is an $n \times m$ matrix, the element a_{ij} $1 \le i \le n$, $1 \le j \le m$ of A, which is the execution time of task t_i on processor p_j.
- D is a linear matrix, the element d_i $1 \le i \le n$ of D, which is the data volume for task t_i to be transmitted when task t_i is to be executed on a remote processor.
- F is a linear matrix, the element f_i $1 \le i \le n$ of F, which is the target processor that is selected for task t_i to be executed on.
- C is a linear matrix, the element c_i $1 \le i \le n$ of C, which is the processor that the task t_i is worked on.
- The processor load for each processor is stated as follow:

$$Load(p_i) = \underbrace{\sum_{j=1} a_{j,i}}_{\substack{No.\ of\ allocated\ processes\ on\ processor\ i}} + \underbrace{\sum_{k=1} a_{k,i}}_{\substack{No.\ of\ New\ Assigned\ processes\ to\ processor.i}} \tag{5}$$

- The *Makespan* of a schedule is the maximal finishing time of all processes or maximum load.

$$makespan(T) = \max\left(Load(p_i)\right) \quad \forall 1 \le i \le Number\ of\ processors \tag{6}$$

- Communication Cost (CC) is computed as follow:

$$CC(T) = \sum_{i=1}^{\substack{number\ of \\ new\ processes}} \left(r_{c_i f_i} + h_{c_i f_i} \times d_i\right) \tag{7}$$

- The Processor utilization for each processor and the average of processors utilization are also computed as follow:

$$U(p_i) = Load(p_i) \Big/ makespan \tag{8}$$

$$AveU = \left(\sum_{i=1}^{No\ of\ processors} U(p_i)\right) \Big/ Number\ Of\ processors \tag{9}$$

- Number of Acceptable Processor Queues (NoAPQ): thresholds for light and heavy load on processor must be defined. If the tasks completion time of a processor is within the light and heavy thresholds, this processor queue will be acceptable. If it is above the heavy threshold or below the light-threshold, then it is unacceptable. But what is important is the average of number of acceptable processors queues, which is achievable by :

$$AveNoAPQ = NoAPQ / Number\ Of\ processors \tag{10}$$

A Queue associated with every processor shows the tasks that processor must execute. The execution order of tasks on each processor is based on queues. Finally, the fitness of the chromosome (Schedule T) can be computed as follow:

$$fitness(T) = \frac{(\gamma \times AveU) \times (\theta \times AveNoAPQ)}{(\alpha \times makespan(T)) \times (\beta \times CC(T))} \tag{11}$$

In the above mentioned formula, $0 < \alpha, \beta, \gamma, \theta \le 1$ are control parameters to control effect of each part according to special cases and their default value is one. This equation shows that a fitter solution (Schedule) has less Makespan, less Communication Cost, higher processor utilization and higher Average number of acceptable processor queues.

Now the proposed method in details is described. Figure 2 depicts the proposed method.

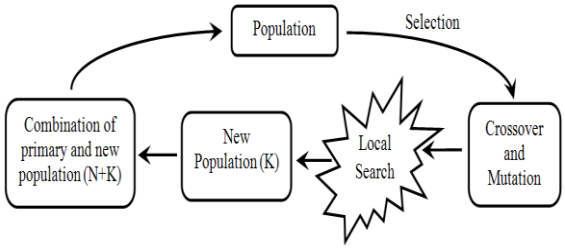

Fig 2: Structure of the proposed method

Bee Colony Optimization (BCO) has been applied as local search in the proposed memetic algorithm. The proposed method works as follows:

In each step of iteration, a task in the chromosome is randomly selected (index of the chromosome) according to probability function, and then the selected task is allocated to its processor. The bee colony optimization is checked to see whether a new chromosome has been found. This function is updated at each iteration in a reinforcing way by using the

features of good solutions. The update of the probability function is done by the bee that produced the best overall chromosome [22].

4. EXPERIMENTAL RESULTS

In this section, regarding Communication Cost, CPU utilization and Makespan, the proposed method has been compared with the existent GA-based method [21], Tabu-Based method[23] and Learning Automata-Based methods[24].

First experiment:

In this experiment attempt has been made to increase the number of tasks and compute average of all CPU utilization, Communication Cost and Makespan. Figures 3 through 5 depict experimental results. As shown in figure 3, increasing the number of tasks, BCO leads to better results compared to three existing methods.

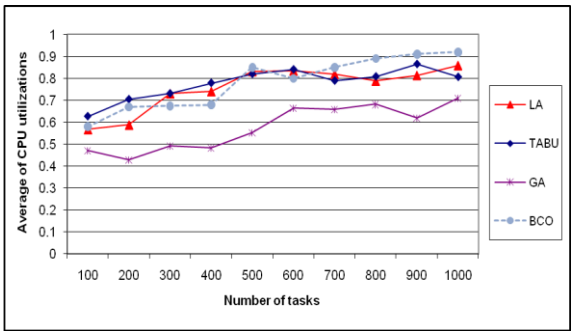

Fig 3: Average of CPU utilization in all methods considering number of task

Figure 4 shows that when the number of the tasks increase, BCO method has approximately less Communication Cost than that of the other methods. But, as it can be seen from figure 5, compared to other methods, BCO shows better results related to Makespan.

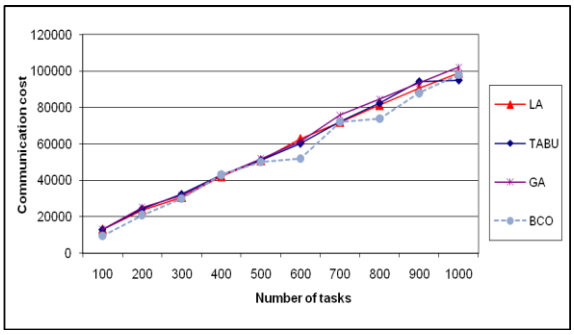

Fig 4: Communication Cost in all methods considering number of task

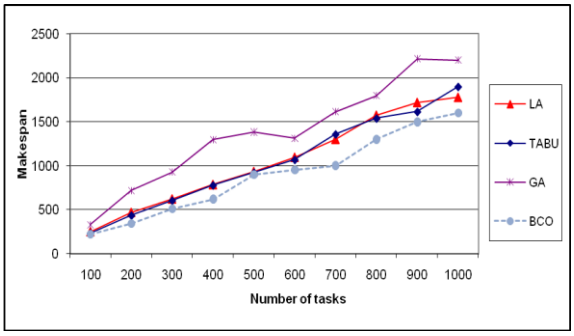

Fig 5: Makespan in all methods considering number of task

Second experiment:

Similarly, regarding an increase in population size, computing the Makespan, Communication Cost and CPU utilization of all processors is the aim of this experiment. According to Figure 6, when the population size increases, Makespan in BCO method is less than that of the other methods.

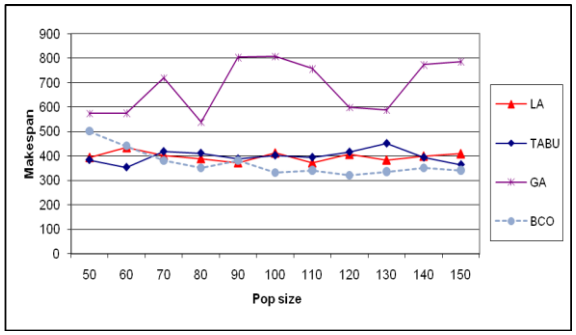

Fig 6: Makespan in all methods regarding pop size

Like Figure 6, figure 7 illustrates that a rise in population size leads to better results in average of CPU utilization in three existing methods. As it can be seen, GA-based method has underutilization.

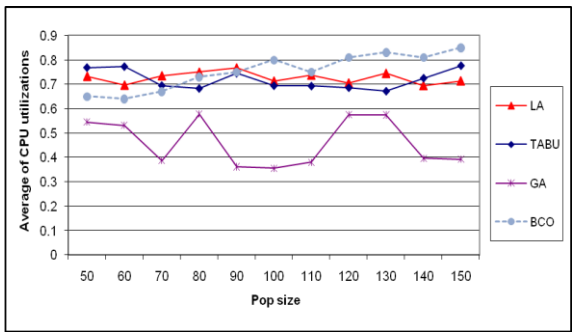

Fig 7: Average of CPU utilization in all methods regarding pop size

It can be observed from figure 8, BCO method has less Communication Cost than that of the other methods.

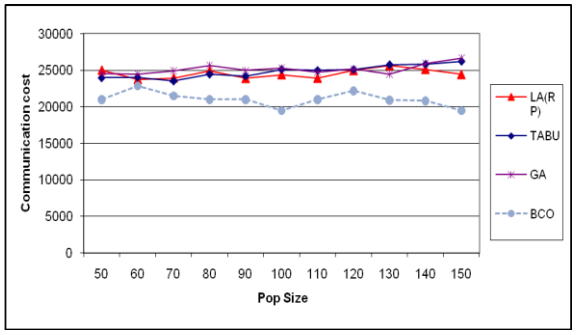

Fig 8: Communication Cost in all methods regarding pop size

Third Experiment:

The objective of this experiment is to compute the above metrics when the numbers of generations are increased. Figures 9 through 11 show the experimental results. Figure 9 shows the average of CPU utilization in all methods. As the number of generations increase, average of CPU utilization in BCO method rises, while the CPU utilization in GA-based method, Compared with other methods, falls. In other words, GA-based method has not scalability.

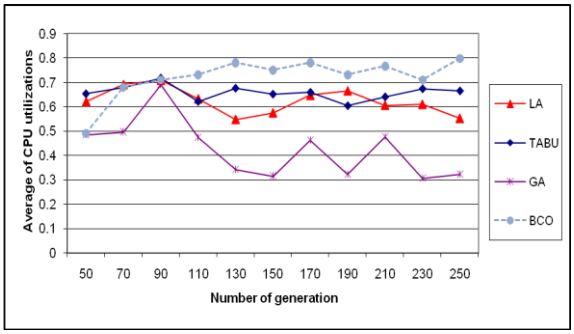

Fig 9: Average of CPU utilization in all methods with respect to number of generation

Figure 10 depicts Communication Cost in all methods. As it can be seen, Communication Cost in BCO method is less than that of the other methods. Also, figure 11 shows that Makespan in BCO method is less than that of the other methods when the number of generation is increased.

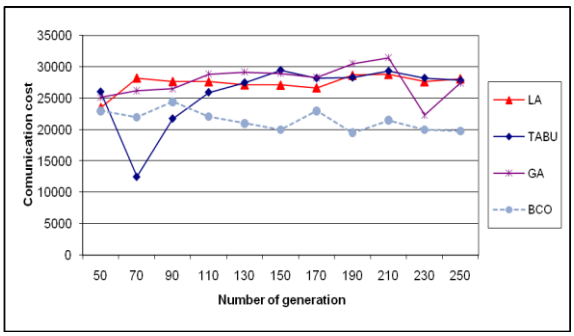

Fig 10: Communication Cost in all methods with respect to number of generation.

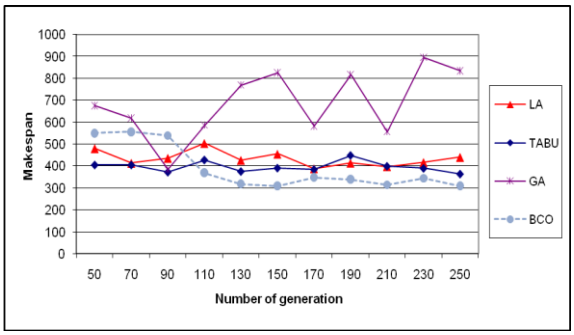

Fig 11: Makespan in all methods with respect to number of generation.

5. CONCLUSION

Scheduling in distributed systems has a significant role in overall system performance and throughput. Scheduling problem is known as NP-complete. In this paper, Memetic algorithm has been used for task scheduling. The author applied Bee Colony Optimization and Ant Colony Optimization as a local search in memetic. This algorithm considers multi objectives in its solution evaluation and solves the scheduling problem; it simultaneously minimizes Makespan and Communication Cost while maximizes the average of CPU utilization. Most existing approaches tend to focus on one of the objectives. Extended experimental results demonstrate that the proposed method outperforms the existent GA-based method in terms of Communication Cost, CPU utilization and Makespan.

6. REFERENCES

[1] Shen, C.C. and Tsai, W.H. 1985. A Graph Matching Approach to Optimal Task Assignment in Distributed Computing Using a Minimax Criterion. IEEE Trans. On Computers, Vol. 34, 197-203.

[2] Ma, P.Y.R., Lee, E.Y. S. and Tsuchiya J. 1982. A Task Allocation Model for Distributed Computing Systems. IEEE Trans. On Computers, Vol. 31, 41-47.

[3] Park, G.L. 2004. Performance Evaluation of a List Scheduling Algorithm In Distributed Memory Multiprocessor Systems. International Journal of Future Generation Computer Systems, Vol. 20, 249-256.

[4] Park, C.I. and Choe, T.Y. 2002. An optimal scheduling algorithm based on task duplication. IEEE Trans. on Computers, Vol. 51, 444–448.

[5] Woodside, C.M. and Monforton, G.G. 1993. Fast Allocation of Processes in Distributed and Parallel Systems. IEEE Trans. On Parallel and Distributed Systems, Vol. 4, 164-174.

[6] Page, A.J., Keane, T.M. and Naughton, T.J. 2010. Multi-heuristic dynamic task allocation using genetic algorithms in a heterogeneous distributed system. Journal of Parallel and Distributed Computing, Vol. 70, 758-766.

[7] Sarje, A.K. and Sagar, G. 1991. Heuristic Model for Task Allocation in Distributed Computer Systems. Proc. of the IEEE, Vol. 138, 313-318.

[8] Chow, Y. and Kohler, W.H. 1979. Models for Dynamic Load Balancing in a Heterogeneous Multiple Processor System. IEEE Transactions on Computers, Vol. 28, 354-361.

[9] Lin, M. and Yang, L.T. 1999. Hybrid Genetic Algorithms for Scheduling Partially Ordered Tasks in a Multi-processor Environment. Proc. of the 6th IEEE Conf. on Real-Time Computer Systems and Applications, 382–387.

[10] Yao, W., Yao, J. and Li, B. 2004. Main Sequences Genetic Scheduling For Multiprocessor Systems Using Task Duplication," International Journal of Microprocessors and Microsystems, Vol. 28, 85-94.

[11] Moore, M. 2003. An Accurate and Efficient Parallel Genetic Algorithm to Schedule Tasks on a Cluster. Proc. of the IEEE International Parallel and Distributed Processing Symposium.

[12] Martino, V.D. 2003. Sub Optimal Scheduling in a Grid using Genetic Algorithms. Proc. of the IEEE International Parallel and Distributed Processing Symposium.

[13] Zomaya, A.Y., Ward, C. and Macey, B. 1999. Genetic Scheduling for Parallel Processor Systems: Comparative Studies and Performance Issues. IEEE Trans. On Parallel and Distributed Systems, Vol. 10, 795-812.

[14] Woo, S.H., Yang, S., Kim, S. and Han, T. 1997. Task scheduling in distributed computing systems with a genetic algorithm. High-Performance Computing on the Information Superhighway, HPC-Asia '97, 301-307.

[15] Hou, E.S.H., Ansari, N. and Ren, H. 1994. A Genetic Algorithm for Multiprocessor Scheduling. IEEE Trans. On Parallel and Distributed Systems, Vol. 5, 113-120.

[16] Lan, Y. and Yu, T. 1995. A Dynamic Central Scheduler Load-Balancing Mechanism. Proc. of the 14th IEEE Ann. Int'l Phoenix Conf. on Computers and Communication, 734-740.

[17] Bonomi, F. and Kumar, A. 1990. Adaptive Optimal Load-Balancing in a Heterogeneous Multiserver System with a Central Job Scheduler. IEEE Trans. on Computers, Vol. 39, 1232-1250.

[18] Teodorovic, D., Davidovic, T. and Selmic, M. 2011. Bee Colony Optimization: The Applications Survey. ACM Transactions on Computational Logic, 1-20.

[19] Karaboga, D. and Basturk, B. 2008. On the performance of artificial bee colony (ABC)," Algorithm. Appl. Soft Comput, Vol. 8, 687–697.

[20] Hanani, A., Nourossana, S., Haj seyed javadi, H. and Rahmani, A. 2010. Solving the Scheduling Problem in Multi-Processor Systems with Communication Cost and Precedence using Bee Colony System. in Proc. of the 3rd International Conference on Advanced Computer Theory and Engineering, V5-464-V5-469.

[21] Nikravan, M. and Kashani, M.H. 2007. A Genetic Algorithm For Process Scheduling In Distributed Operating Systems Considering Load balancing. in Proceedings of the 21th European Conference on Modeling and Simulation, 645-650.

[22] Kashani, M.H., Jamei, M., Akbari, M. and Moosavi Tayebi, R. 2011. Utilizing Bee Colony to Solve Task Scheduling Problem in Distributed Systems. in Proceedings of the Third International Conference on Computational Intelligence, Communication Systems and Networks, 298-303.

[23] Jahanshahi, M., Gholipour, M., Kordafshari, M.S. and Rahmani, A.M. 2009. A Novel Method for Task Scheduling in Distributed Systems Using Memetic. in proceeding of the Second International Conference on Communication Theory, Reliability, and Quality of Service, 58-62.

[24] Jahanshahi, M., Meybodi, M.R. and Dehghan, M. 2009. A New Approach for Task Scheduling in Distributed Systems Using Learning Automata. Proceedings of the IEEE International Conference on Automation and Logistics Shenyang, 62-67.