Biogeography-Based Optimization of Makespan and Reliability in Grid Computing Systems

Mohammad Hadi Mobini, Reza Entezari-Maleki, and Ali Movaghar
Performance and Dependability Laboratory, Department of Computer Engineering, Sharif University of Technology, Tehran, Iran.
{mobini, entezari}@ce.sharif.edu, movaghar@sharif.edu

***Abstract*—The aim of this paper is to propose a scheduling method to consider reliability along with makespan in grid computing systems. The reliability of task execution is considered in the proposed method to increase the chance of running large-scale and computationally intensive workflows successfully. To handle situations in which a resource failure in a possible scheduling solution occurs, the proposed method finds a collection of scheduling solutions instead of only one solution to run the workflow. It leads to have chance to run an alternative scheduling solution that is not using the failed resource. To find the most optimized scheduling solutions, we exploit the lately developed biogeography-based optimization method with evaluation strategy and combine it with the operations like neighborhood search and crossover. Finally, the proposed method is compared with two successive scheduling methods. The results obtained from simulations show that gained improvement is significant especially in large workflows with large number of tasks.**

***Index Terms*—Heterogeneous networks, grid computing, execution time, reliability, biogeography-based optimization.**

Notations

r_i	resource *i*
v_i	task *i*
s_i	scheduling *i*
Time(s)	finish time of scheduling *s*
Fail(s)	fail factor of scheduling *s*
λ_i	failure rate of resource *i*
$R(i,\Delta t)$	reliability of resource *i* in period Δt
p_i	computation power of resource *i*
N	number of tasks
P	number of computational resources
$Childs(v_i)$	tasks in DAG that v_i is their predecessor
$Parents(v_i)$	tasks in DAG that are predecessors of v_i
$subsets(s)$	all subset solutions of *s* in related collection of scheduling solutions
$Pr(s)$	the probability of successful execution of scheduling *s*
BusyResources(s)	all computational resources in scheduling *s* allocated to execute tasks
$Pr(\bar{s} \cap \bar{r})$	failure probability of solution *s* only because of resource *r* failure
t_i^s	first feasible start time of executing task *i*
t_j^f	first feasible finish time of executing task *i*
$O(v_i)$	the priority of task v_i

I. Introduction

Distributed computing systems make it possible to bring more flexibility in large scale computing using various kinds of software and hardware platforms [1]. In many scientific and industrial fields, it is needed to use large-scale computing systems to solve very huge and time-consuming problems. In recent years, grid computing as a major part of distributed computing systems is extremely used by scientific and industrial organizations to solve the problems [2, 3]. Using grids is a powerful way of making execution of large processing workflows possible with much lower cost than using supercomputers.

While running a large-scale workflow, users are interested to use methods to make the job done as early as possible. Despite this effort to shortening the executing time of workflows, there are still many cases that require the distributed system to work hours and even days to complete the job. In these cases, in addition to the time importance, the reliability of computing resources is an important issue, as well [4, 5]. The resource reliability becomes more important when execution of a job takes more time. In such cases, even a single failure in one of active computing resources can make the entire execution process failed. Hence, reliability of computing resources should be considered by system developers to construct more dependable systems and provide users with the fast and reliable services.

Since workflow jobs can be divided into many tasks that may have data or control dependencies to each other, finding the best possible scheduling of tasks to run on available computing resources can be an interesting problem. Moreover, how to dispatch the tasks among the available resources is very important problem influencing total execution time of the job and reliability of job execution. In order to appropriately dispatch the tasks among the resources, topology of the network should be considered. Each distributed system has a network topology representing the location of the resources and relationship between all resources and links. Star and tree topologies are more famous topologies considered by many researches to show the communication status of the resources in grid environments [6, 7, 8]. In many of these researches, grid computing has a head, named resource management system (RMS) [8, 9]. Figure 1 shows a sample tree topology of the grid computing environment. In our study, RMS is responsible for sending the tasks produced by splitting a single workflow job to computing resources and monitoring the resources to know if they fail to do their own work [8, 9].

In this paper, a method to consider the reliability and performance of workflow applications in grid computing environment is presented. The method uses the recently developed biogeography-based optimization (BBO) approach with evolutionary strategy algorithm [10, 11]. Also, it is assumed that the topology of the environment is tree and RMS is in the root of the network, because this kind of network topology is one of the well-known and realistic topologies in many grid computing systems [8]. The primary novelty of the proposed method is, instead of finding just one scheduling solution to assign workflow tasks to computational resources, finding a collection of scheduling solutions to be used alternatively when a resource failure leads to impossibility of executing one of proposed scheduling solutions.

The remaining parts of the paper are organized as follows. Section II introduces some research works previously done on reliability and performance optimization of distributed computing networks. Section III describes background information that partially related to the problem. In section IV, the proposed method of scheduling is introduced in details. In section V, the proposed algorithm is compared against other recently developed algorithms. Finally, section VI concludes the paper and discuses about future works which can be done in this field of research.

II. Related Works

Makespan optimization is almost a classic problem in grid or distributed computing environments [2, 12, 13, 14]. Furthermore, the problem of finding scheduling solution with the minimum makespan is a famous NP-hard problem [9, 15, 16, 17]. Therefore, using heuristic methods to solve this problem is inevitable and many researches in this context have been done recently [7, 9, 17, 18, 19, 20].

Entezari-Maleki et al. [19] have proposed a genetic based task scheduling algorithm to minimize the makespan of grid applications. The algorithm proposed in [19] and other aforementioned researches [7, 9, 17, 18, 19, 20] only concentrate on minimizing makespan, but as in large-scale processing workflows reliability of computational resources is a basic principle which must be considered to be able to propose applicable scheduling solutions for that kind of workflows.

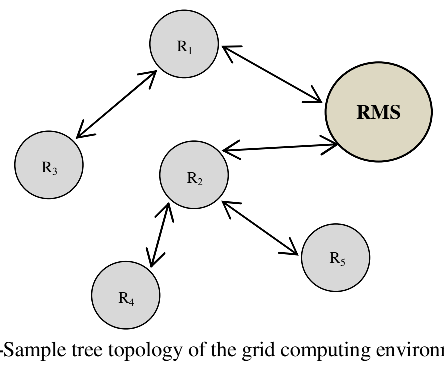

Fig. 1–Sample tree topology of the grid computing environment

In order to simultaneously consider the reliability and makespan measures in proposing scheduling solutions, it is required to offer a balance model, because minimizing makespan and maximizing reliability are two inconsistent objectives [9, 17, 21]. To handle the inconsistency existing between these two objectives, Dai et al. [8] proposed a deadline-based model which first generates all feasible scheduling solutions with makespan less than a predefined deadline, and then finds the best possible solution with the maximum reliability between feasible solutions. The model can only be applied to the workflows with few tasks and systems with few computational resources because it is generating all feasible solutions which may lead to long processing time when applying to the larger workflows or grid systems. Therefore, the model proposed in [8] is not well designed to be used by heuristic methods those can find better solutions with much less processing time. Hence, it is better to use a modified version of an acceptable balance model proposed in [9].

Attiya el al. [17] have proposed a cost based reliability model which is only used to maximize reliability and not considering makespan, this trend may simply end in high reliability scheduling solutions with very long makespan. Moghaddam et al. [20] have used a fitness function to balance between makespan and cost of executing each scheduling solution. The same trend can be used to balance between reliability and makespan like the proposed method of Xiaofeng et al. [9]. In [9] the fitness function is like what is used in [20] but instead of execution cost the reliability is considered. This kind of balance model can be used by heuristic methods [9, 16]. Hence, we use a modified version of balance model proposed in [9]. The model is described in section IV with details.

Various kinds of heuristic methods like list heuristics [22] and genetic algorithms (GAs) [9, 16, 20] have been used for this multi-objective optimization problem. All the proposed heuristics are designed to optimize a single scheduling solution not a collection of solutions. Therefore, we use lately developed biography-based optimization with evaluation strategy [10, 11] to find a collection of scheduling solutions.

III. Background Information

In this paper, a job workflow consisting of many tasks with data and control dependencies between the tasks is considered. It is assumed that a job is modeled with a directed acyclic graph (DAG) which shows data or control dependencies among the tasks. A sample DAG is depicted in Fig. 2.

As shown in Fig. 2, data or control dependencies are illustrated with directed lines between the nodes representing tasks. In this paper, we point each task with $v_{i,}$ Moreover, each task may vary in its computational complexity. The computational complexity of task i, $|v_i|$, is the number of instructions required to be executed to process that specific task, so it can be represented with an integer number.

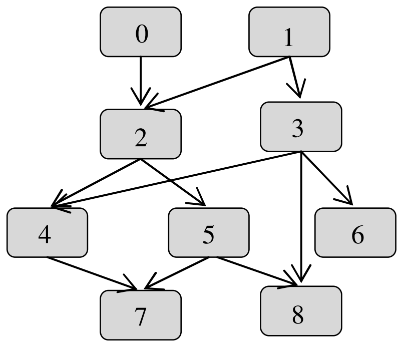

Fig. 2–A sample DAG

Furthermore, each task may have zero or more parents in the workflow DAG. If it has no parents, the task is an entry task and if it is not parent of any task, it is an exit task. It is obvious in Fig. 2 that a DAG may have many entrance or exit nodes.

In a heterogeneous computer network, each computation resource, r_i, has its own independent computation power, p_i. The power of a computation resource is the number of instructions which can be executed by that resource in each second. Therefore, to compute the execution time of a task on a computation resource, one can divide the computation complexity of the task by power of that resource.

Other important characteristic of a computation resource is its failure rate which is independent from other computation resources in network just similar to its computation power [2, 15]. The reliability of the resource r_i is important because it can be used to calculate the probability of failure in a period of time, Δt, using exponential distribution [8, 23].

$$R(i, \Delta t) = e^{-\lambda_i \Delta t} \tag{1}$$

Now, the execution time of a task and reliability of executing that task on a specific resource can be calculated. Since, minimizing the execution time of a task and simultaneously maximizing the execution reliability is a NP-hard problem [8, 9, 13, 14, 15], we can formulize the problem and appropriately use heuristic methods to find the fittest possible scheduling.

IV. The Proposed Scheduling Method

In this section, the model to calculate the reliability and fitness of each possible scheduling solution is presented in the first, and then the proposed scheduling method is presented based on the reliability calculation model.

IV-A. *Reliability Model*

The reliability of a specific scheduling is the probability in which all resources in that scheduling stay operational and no failure occurs during executing jobs [8, 9, 22, 24].

To achieve the goals of this paper, maximizing scheduling reliability and minimizing total execution time, defining a fitness function to compare the suitability of a scheduling with the others is required. The functions *F(s)* is giving every scheduling solution rank between zero and one and the better scheduling has rank closer to one [9]. Functions *F(s)* is shown in Eq. 2.

$$F(s) = \omega_1 . \frac{Fail(s) - MinFail}{MaxFail - MinFail} + \omega_2 . \frac{Time(s) - MinTime}{MaxTime - MinTime} \tag{2}$$

$$Fail(s) = \sum_{i=0}^{P} t_i \lambda_i \tag{3}$$

where t_i is the time in which resource i finishes its own work of executing all tasks assigned to it. Moreover, *MaxFail* and *MinFail* are empirical values obtained by observing experiences in environment and are used to normalize the result of *Fail* function to be valid aggregating it with the result of *Time* function in other half of formula.

Time(s) is the time to finish executing all tasks existing inside a workflow. This measure is defined as Eq. 4.

$$Time(s) = Max(t_i),\ 0 < i < N \tag{4}$$

Like first half of Eq. 2, the time part also needs to be normalized to be valid aggregating it with other half. So, *MaxTime* and *MinTime* can be found by RMS via monitoring the entire grid environment.

Variables ω_1 and ω_2 are preference factors chosen by user as his/her favorite of reliability or makespan importance.

IV-B. *The Details of the Proposed Method*

In our approach, it is assumed that if a resource fails it cannot be recovered. Therefore, every scheduling solution containing the failed resource will be failed. In order to handle this kind of failure, the idea of using a collection of scheduling solutions can be used instead of proposing only one scheduling solution [8]. In [8], all possible scheduling solutions which have makespan less than predefined deadline are used, but we use *log(P)* as number of final proposed scheduling solutions as result of our scheduling method. Using this idea, we can usually have at least one alternative scheduling solution for a single resource failure and have chance for more than one resource failure.

To calculate the fitness of a collection of scheduling solutions, *C*, the expected value is used. First, all of the available scheduling solutions are sorted by their number of free resources descending. The number of free resources in each solution is number of resources with no assigned task in that specific scheduling solution. Therefore, s_0 has more free resources than s_1 if both of them are in the same collection. Moreover, busy resources are those who have at least one assigned task in the related scheduling solution.

We used a recursive method to calculate total fitness of a collection of solutions. To explain this method it is required to first define some concepts.

Lemma 1. Solution *A* is a subset of solution *B* if and only if every free resource in *A* is free in *B*, as well. With this definition of subset solution, one can assure that failure of a free resource in superset solution cannot lead to failure in subset solution.

Lemma 2. Fitness of a collection of scheduling solutions is defined as maximum expected fitness of its solutions and subsets of each solution.

To use *lemma 2* it is required to define expected fitness of a solution and its subsets, $\acute{F}(s)$.

$$\acute{F}(s) = \begin{cases} F(s) \times \Pr(s), & if\ subsets(s) = \varphi \\ F(s) \times \Pr(s) + \sum_{i=1}^{size\ of\ subsets(s)} \max\left(\acute{F}(s_i) \times \Pr(\bar{s} \cap \bar{r} | s_i)\right); & \\ \forall r \in busyResources(s), & otherwise \end{cases} \quad (5)$$

where $subsets(s)$ denotes all of the solutions existing in the same collection containing s and can be considered as subsets of s according to *lemma 1*. Also $Pr(s)$ means the probability of occurring no failure in busy resources of solution s during its total execution time. $Pr\ (\bar{s} \cap \bar{r} | s_i)$ is the probability of failure of resource r which leads to failure of solution s while no other busy resource of s fails during its total execution time. Figure 3 shows the algorithm used to calculate the fitness of a collection of solutions.

When a collection of scheduling solutions is selected as a final result of proposed algorithm, we can deliver all solutions to the user or just the solution which has maximum $\acute{F}(s)$ along its subsets in that collection.

As mentioned before, finding the fittest solution for such problem is a NP-hard problem, so we use a heuristic method to solve the problem. To be able to use heuristic methods it is required to represent each scheduling solution by a simple data structure. Each scheduling solutions consists of two parts; the first part is assignment of each task to a computational resource and the second one is the order of executing tasks which are assigned to one computational resource. In order to present the first part, an array of integer numbers, like what is illustrated in Fig. 4, is used. This kind of arrays are called solutions chromosome.

In each chromosome, the number corresponding to a location shows the resource in which allocated to process that task. As an example, the sample chromosome represented in Fig. 4 implies that resource r_4 is allocated to process tasks t_0 and t_2.

Since the order of executing the tasks which are scheduled on a computational resource is not covered in chromosome, we should define a new function named Order function. This function can be seen in Eq. 6.

```
**fitnessCalculator** (solution collection $C$) {
  sort decreasingly all solutions in $C$ by number of free resources.
  **for** each $s$ in $C$
    calculate $\acute{F}(s)$ for all sorted solutions and store it for future use.
  **end for**
  **return** the maximum $\acute{F}(s)$.
}
```

Fig. 3–Algorithm to find fitness of a collection of scheduling solutions

Task ID	0	1	2	3	4	5	6	7	8	9
Resource	4	1	4	5	8	1	2	3	5	1

Fig. 4–A sample chromosome

$$O(v_i) = \begin{cases} |v_i|, & Childs(v_i) = \varphi \\ |v_i| + \max\left(O(v_j)\right), & v_j \in Childs(v_i) \end{cases} \quad (6)$$

where $O(v_i)$ is the sum of size of tasks in the longest path beginning from task i in the related DAG. Also, the output of function $Childs(v_i)$ is the collection of tasks in which v_i is their parent in the related DAG. Therefore, $O(v_i)$ can be used to determine the order of execution between tasks which are scheduled to run on a computational resource. A task with higher $O(v_i)$ will be executed earlier.

To consider the data and control dependency of tasks, it is required to adjust execution start time of each task by execution finish time of its parents in DAG.

$$t_i^s = \begin{cases} 0, & Parents(v_i) = \varphi \\ Max\left(t_j^f\right), & v_j \in Parents(v_i) \end{cases} \quad (7)$$

where t_i^s is the first time in which the task v_i can be executed by any computation resource and t_i^f is the first possible finish time of task v_i on the related scheduling solution. Moreover, $Parents(v_i)$ shows a collection of parent nodes of task v_i in the related DAG. While we respect finish time of parents in defining t_i^s, it can help us to be sure that all predecessors of a given task are executed before that task.

The heuristic method used in this paper to find the best possible solution is a meta-heuristic algorithm based on biogeography-based optimization (BBO) which is a new kind of heuristic algorithms and has already outperformed older proposed heuristic methods in many famous problems [10, 11].

The basic idea of BBO is making groups of solutions that each group of solutions represents an island. In every iteration, solutions can immigrate to other islands and make new groups. With this method, solutions which have better fitness functions can immigrate to the groups with higher fitness to make new islands with even higher fitness. In addition, it is possible to migrate solutions with lower fitness to other islands and make well ranked islands more pure. In the following, each of the steps used in our proposed approach are listed and explained by details.

Crossover and Selection

In each iteration of our proposed method, only exchanging solutions between the groups is not enough, because it may result in sticking to specific part of solution space and not finding the best global scheduling solutions possible. Hence, it is required to generate new solutions randomly to attenuate chance of sticking in small part of solution space.

One of the most efficient available methods in bringing randomness into proposed method is the use of crossover on two different solutions to see if the result is even better or not. Therefore, we implement cut-off based crossover method which randomly selects an index in chromosome and exchange left and right parts of the chromosomes to generate two different scheduling solutions. In such crossover process, we only exchange task-resource assignments and postpone finding the order of execution of tasks to the evaluation phase. This means, when we want to calculate the fitness of a scheduling solution before using fitness formula (Eq. 4) the algorithm will find best order of task execution of each resource using Eq. 6. A sample crossover operation is shown in Fig. 5.

Before using crossover to make new random scheduling solutions, we use roulette wheel based method to select half of top solutions in each group, and then apply crossover to some random pairs to make new group with the same population. It is also possible to first double the population size of group using crossover, and then use roulette wheel method to filter solutions in a new group and select half of them. In our experiences, the first method usually results better group fitness.

While new chromosomes are generated during crossover, we can use the lemma proved by Dongarra et al. [21] to make better results in crossover phase. The lemma is used to refuse or accept a task assignment change. As proved in [21], the resource with less proportion of resource failure rate to resource power is better to be allocated to a task. Hence, we can filter changes in crossover using this lemma and only accept resource assignments which are going to make better solutions.

Neighborhood search

Neighborhood search is an operation which is applied to random number of scheduling solutions in each group to replace a selected scheduling solution with its local optimum solution. To find local optimum of specific solutions, it is required to try all resources for each of tasks in that solution, and finally accept the fittest neighbor of that solution. Since this operation is time consuming and its time complexity is $O(P \times N)$, we only apply it to random number of solutions. Local search algorithm is shown in Fig. 6.

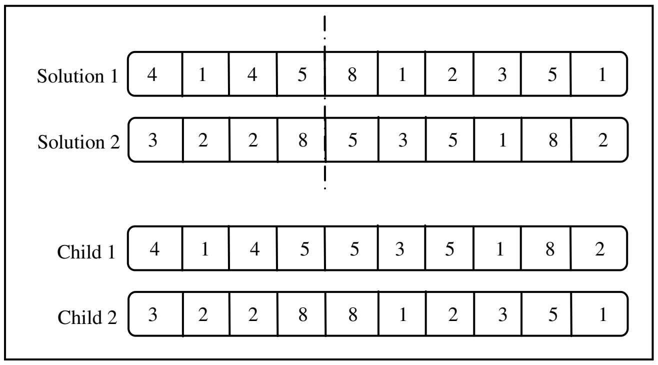

Fig. 5–Cut-off based crossover sample

```
neighborhoodSearch (solution S){
  Define $S_0 = S$
  for each v in S
        for each r in all resources
              assign v to r and make new solution $S_1$
              if $\acute{F}(S_1) > \acute{F}(S_0)$ then
                    $S_0 = S_1$
              end if
        end for
  end for
  return $S_0$
}
```

Fig. 6–Neighborhood search algorithm

V. Experimental Results

In order to assess the efficiency of the proposed scheduling method, we simulate the method and compare it with other similar algorithms. Since there is no standard benchmark to compare proposed method, wide range of randomly generated data have been used in our experiments. This method has been used in many research works [8, 9, 16, 17]. We use random DAG generator [16, 17] to generate a random DAG. The parameters which DAG generator accepts as its input are: population size, minimum and maximum out degree of each node in DAG, minimum and maximum computation complexity. The DAG generator uses uniform distribution to generate random numbers. To obtain more dependable results, the randomly generated DAG is used for all compared algorithms for eight times and the average value of each of the gained results is used for comparison.

In each iteration of the proposed method as well as a process dependency DAG, information about number of computation resources, their computation powers and failure rates are required. Therefore, we generate a list of random resources using number of resources, mean and deviation of computation power, mean and deviation of failure rate as input parameters. The normal distribution is used to generate random numbers in this part. Table 1 shows the value of aforementioned parameters used in the experiments.

Assumptions about the workflow job and grid system considered in our experiments are as follows.

- The entire grid system is monitored by RMS which is responsible to find the best solution and dispatch the tasks to the resources.
- RMS is connected to all resources directly or indirectly via other resources.
- The types of workflow jobs considered in this paper are mostly computation intensive, so the communication links between the resources are not considered in reliability nor makespan evaluation.
- Each computation resource can only execute a task at once; therefore, each resource has its own waiting list.
- The RMS is aware of computation power and failure rate of all resources.
- Once a resource fails due to hardware or software problems, it is not possible to be recovered, because the subjection of our study is mainly related to reliability evaluation while considering resource recovery is used in availability assessment [25].
- In all of the experiments, without loss of generality, preference values have been set to 0.5 ($\omega_1 = \omega_2 = 0.5$). These values only show the preference of user and can be set to the other values, as well.

Table 1–Resource specifications in four different scenarios

Senario	Mean failure	Failure Deviation	Mean Power	Power Deviation
1	0.2	0.15	20	10
2	0.01	0.0015	20	10
3	0.2	0.15	200	20
4	0.01	0.0015	200	20

To show the efficiency of the proposed method, biogeography-based optimization with evaluation strategy (BBOES), we have implement two successful recently developed scheduling algorithms, honeybee-mating optimization (HBMO) [16] and look ahead genetic algorithm (LAGA) [9], and compared our method with these two algorithms. Figures 7 to 10 compares the proposed algorithm (BBOES) with HBMO and LAGA while *senario1* to *senario4* are considered. As can be seen in Fig. 7, the BBOES outperforms both of the algorithms in respect of normalized fitness. In Fig. 7 to Fig. 10, the vertical axes show normalized rank of fitness for each algorithm; therefore, the higher rank is the better one. The horizontal axes are for ratio of number of task to number of resources which starts from 0.5 and ends with 16.

As shown in Fig. 7 to Fig. 10, the proposed algorithm leads to major improvement on problem of optimizing both makespan and reliability of heterogeneous computing networks in most cases. However, as the ratio of number of tasks to number of computation resources increases the improvement of proposed method over LAGA and HBMO becomes more significant.

VI. Conclusion and future work

In large-scale computation intensive workflows, scheduling tasks on computational resources to minimize the makespan due to possibility of failure in computational resources is not enough, because failure in a computational resource may lead to failure of whole execution process. Moreover, as minimizing makespan and maximizing reliability are two inconsistent objectives a balance model can be used to define a fitness function which represents both reliability and makespan characteristics of each scheduling solution. Using the balance model and fitness function enables the ability of using heuristic methods to find most near optimal scheduling solutions. In this work, we have proposed a BBOES based method to solve the problem of optimizing both makespan and reliability in grid networks. The proposed method compared with two other heuristics, LAGA and HBMO algorithms, and the results obtained from simulations show that the improvement is significant in almost all cases. Therefore, the proposed method can be considered as an interesting alternative to what already is in use.

For future works, one can extend the proposed algorithm to find optimized solutions for multi-objective problems; objectives like costs of task execution on different computational resources or availability of resources. In addition, it is possible to consider network reliability and communication time in the reliability model to extend the usability domain of the algorithm covering data intensive workflows.

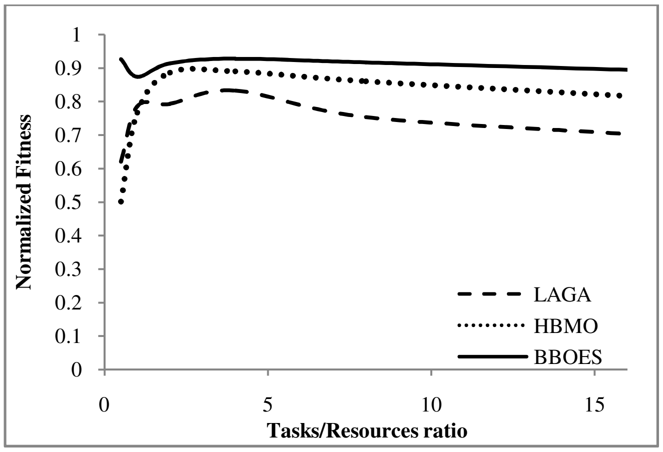

Fig. 7–Comparison of BBOES, LAGA and HBMO at *scenario1*

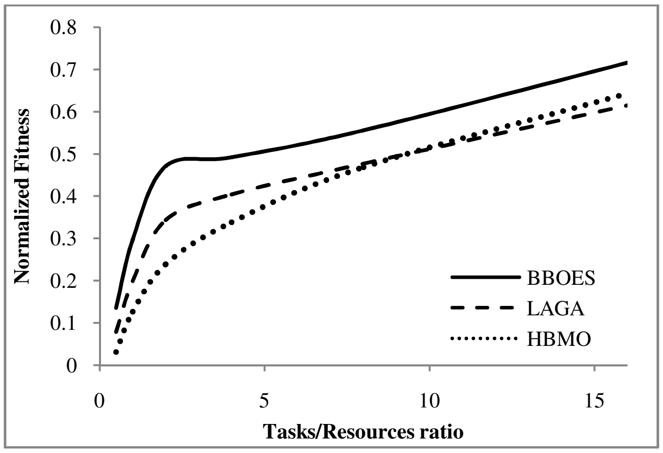

Fig. 8–Comparison of BBOES, LAGA and HBMO at *scenario2*

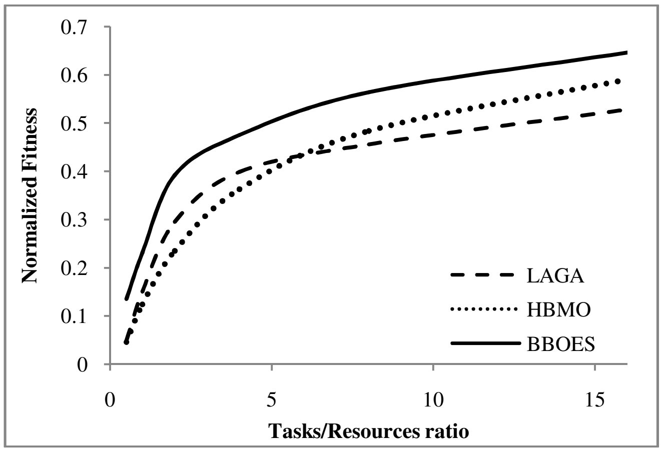

Fig. 9–Comparison of BBOES, LAGA and HBMO at *scenario3*

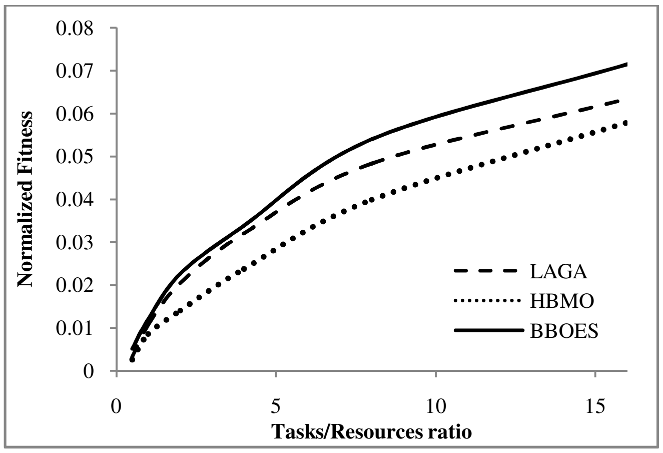

Fig. 10–Comparison of BBOES, LAGA and HBMO at *scenario4*

ACKNOWLEDGMENT

The authors want to express their gratitude to the Iran's National Elite Foundation for their financial support of this research.

REFERENCES

[1] I. Foster, Y. Zhao, I. Raicu, and S. Lu, "Cloud computing and grid computing 360-degree compared," Grid Computing Environments Workshop, Texas, USA, November 12–16, 2008, pp. 1–10.

[2] E. Deelman, et al., "Pegasus: Mapping Scientific Workflows onto the Grid," in Grid Computing, ser. Lecture Notes in Computer Science, M.D. Dikaiakos, Eds., vol. 3165. Springer, January 28–30, 2004, pp. 131–140.

[3] M. Ghanem, N. Azam, M. Boniface, and J. Ferris, "Grid-enabled workflows for industrial product design," Second IEEE International Conference on e-Science and Grid Computing, Amsterdam, Netherlands, December 4–6, 2006, pp. 96.

[4] C. Dabrowski, "Reliability in grid computing systems," Concurrency and Computation: Practice and Experience, vol. 21, no. 8, pp. 927–959, 2009.

[5] Y.S. Dai, M. Xie, and K.L. Poh, "Reliability analysis of grid computing systems," IEEE Pacific Rim International Symposium on Dependable Computing, Tsukuba, Japan, December 16–18, 2002, pp. 97–104.

[6] M. Baker, R. Buyya, and D. Laforenza, "Grids and grid technologies for wide-area distributed computing", Software: Practice and Experience, vol. 32, no. 15, pp. 1437–1466, 2002.

[7] M.S. Chang, "The distributed program reliability analysis on star topologies", International Conference on Parallel and Distributed Systems, Tainan, Taiwan, December 14–16, 1998, pp. 100–106.

[8] Y.S. Dai and G. Levitin, "Performance and reliability of tree-structured grid services considering data dependence and failure correlation," IEEE Transactions on computers, vol. 56, no. 7, pp. 925–936, 2007.

[9] X. Wang, C.S. Yeo, R. Buyya, and J. Su, "Optimizing the makespan and reliability for workflow applications with reputation and a look-ahead genetic algorithm," Future Generation Computer Systems, vol. 27, no. 8, pp.1124–1134, 2010.

[10] D.W. Du, D. Simon, and M. Ergezer, "Biogeography-based optimization combined with evolutionary strategy and immigration refusal," The IEEE Conference on Systems, Man, and Cybernetics, Texas, USA, October 11–14, 2009, pp. 1023–1028.

[11] D. Simon, "Biogeography-based optimization," IEEE Transactions on Evolutionary Computation, vol. 12, no. 6, pp. 702–713, 2008.

[12] E. Bampis, C. Delorme, and J.C. König, "Optimal schedules for d-D grid graphs with communication delays", Parallel Computing, vol. 24, no. 11, pp. 1653–1664, 1998.

[13] S. Parsa and R. Entezari-Maleki, "A queuing network model for minimizing the total makespan of computational grids," Computers and Electrical Engineering, vol. 38, no. 4, pp. 827–839, 2012.

[14] S. Parsa and R. Entezari-Maleki, "Task dispatching approach to reduce the number of waiting tasks in grid environments," The Journal of Supercomputing, vol. 59, no. 1, pp. 469–485, 2012.

[15] M.R. Garey and D.S. Johnson, "Computers and intractability: A Guide to the Theory of NP-completeness," Freeman, San Francisco, 1979.

[16] Q.M. Kang, H. He, H.M. Song, and R. Deng, "Task allocation for maximizing reliability of distributed computing systems using honeybee mating optimization," The Journal of Systems and Software, vol. 83, no. 11, pp. 2165–2174, 2010.

[17] G. Attiya and Y. Hamam, "Task allocation for maximizing reliability of distributed systems: a simulated annealing approach", Journal of Parallel and Distributed Computing, vol. 66, no. 10, pp. 1259–1266, 2006.

[18] Z. Mousavinasab, R. Entezari-Maleki, and A. Movaghar, "A bee colony task scheduling algorithm in computational grids," The International Conference on Digital Information Processing and Communication, Communications in Computer and Information Science, vol. 188, Springer press, Ostrava, Czech Republic, July 7–9, 2011, pp. 200–210.

[19] R. Entezari-Maleki and A. Movaghar, "A genetic-based scheduling algorithm to minimize the makespan of the grid applications," Grid and Distributed Computing Conference, Communications in Computer and Information Science, vol. 121, Springer press, Jeju Island, South Korea, December 13–15, 2010, pp. 22–31.

[20] S. Kardani-Moghaddam, F. Khodadadi, R. Entezari-Maleki, and A. Movaghar, "A hybrid genetic algorithm and variable neighborhood search for task scheduling problem in grid environment," Procedia Engineering, vol. 29, pp. 3808–3814, 2012.

[21] J.J. Dongarra, E. Jeannot, E. Saule, and Z. Shi, "Bi-objective scheduling algorithms for optimizing makespan and reliability on heterogeneous systems" The 19th Annual ACM Symposium on Parallel Algorithms and Architectures, New York, USA, January 2007, pp.280–28.

[22] M. Hakem and F. Butelle, "Reliability and scheduling on systems subject to failures", The International Conference on Parallel Processing, Xi-An, China, September 10–14, 2007, pp. 38.

[23] R. Entezari-Maleki and A. Movaghar, "A probabilistic task scheduling method for grid environments," Future Generation Computer Systems, vol. 28, no. 3, pp. 513–524, 2012.

[24] Y.S. Dai, M. Xie, K.L. Poh, and G.Q. Liu, "A study of service reliability and availability for distributed systems," Reliability Engineering and System Safety, vol. 79, no. 1, pp. 103–112, 2003.

[25] R. Entezari-Maleki and A. Movaghar, "Availability Modeling of Grid Computing Environments Using SANs," The 19th International Conference on Software, Telecommunications and Computer Networks, Split, Croatia, September 15–17, 2011, pp. 1–6.