A Genetic Algorithm to Increase the Throughput of the Computational Grids [1]

Reza Entezari-Maleki, Ali Movaghar
Department of Computer Engineering, Sharif University of Technology, Tehran, Iran
entezari@ce.sharif.edu, movaghar@sharif.edu

Abstract

High throughput computing (HTC) is of great importance in grid computing environments. HTC is aimed at minimizing the total makespan of all of the tasks submitted to the grid environment in long execution of the system. To achieve HTC in grids, suitable task scheduling algorithms should be applied to dispatch the submitted tasks to the computational resources appropriately. In this paper, a new task scheduling algorithm is proposed to assign the tasks to the grid resources with goal of minimizing the total makespan of the environment. The proposed algorithm uses genetic approach to find the most suitable match between the tasks and grid resources. The simulation results obtained from applying the proposed algorithm to schedule independent and sequential tasks to the grid resources demonstrate the applicability of the algorithm in grid environments.

Keywords: *Task scheduling algorithm, grid computing environment, total makespan, genetic approach.*

1. Introduction

There are many computation and data intensive problems in science and industry those require weeks or months of computation to solve. Scientists involved in these types of problems need a computing environment that deliver large amounts of computational power over a long period of time. Such an environment is called a high throughput computing (HTC) environment [1, 2, 3]. The HTC field is more interested in how many independent tasks can be completed over a long period of time instead of how fast an individual task can complete which is interested in high performance computing (HPC) [4, 5]. HPC systems are mostly used to execute parallel and dependent tasks and therefore they must execute within a particular site with low latency interconnections. Conversely, in the HTC systems there are mostly numbers of independent and sequential tasks which can be individually scheduled on many different computing resources across multiple administrative domains. Considering the definition and properties of the HTC systems and comparing them with grid computing [6] concepts, one can easily understand the relation between these two technologies. Computational grids are mainly aimed at supplying high computational power to solve the time consuming and computationally intensive problems in a timely manner. Therefore, grid computing environments try to provide high throughput environments for their users.

In order to achieve the HTC through computational grids, the overall response time to all the tasks, in a relatively long period of time, should be minimized. Thereby, the grid manager should schedule the submitted tasks to appropriate grid resources, considering the total

[1] The brief version of this paper has been presented in the Grid and Distributed Computing Conference (GDC 2010), Jeju Island, South Korea, December 13-15, 2010.

makespan of the environment. The makespan of a resource is the time slot between the start and completion of a sequence of tasks assigned to that resource [7, 8, 9, 10]. The total makespan of a grid computing environment is defined as the largest makespan of the grid resources. Minimizing the total makespan of a grid environment, the throughput of the environment is increased accordingly [4, 7].

To make effective use of the tremendous capabilities of the computational resources distributed within grid environments and minimize the makespan of the grids, efficient task scheduling algorithms are required. Task scheduling algorithms are commonly applied by the grid manager to optimally dispatch tasks to the grid resources [4, 7, 11]. Typically, grid users submit their own tasks to the grid manager to take full advantage of the grid facilities. The grid manager in a computational grid tries to distribute the submitted tasks among the grid resources in such a way that the total response time is minimized. There are several task scheduling algorithms aimed at minimizing the makespan of the distributed systems [4, 7, 10, 12, 13]. Indeed, these algorithms try to minimize the overall completion time of the tasks submitted to the system by finding the most suitable resources to be allocated to the tasks. It should be noticed that minimizing the overall completion time or total makespan of the tasks does not necessarily result in the minimization of execution time of each individual task [11].

Decision about the assigning of tasks to the resources and finding the best match between the tasks and resources is an NP-complete problem [4, 8, 9, 11]. For this reason, some heuristic methods have been proposed to find the suitable and near optimal solutions for scheduling problem [7, 12, 14, 15]. This paper proposes a new task scheduling algorithm to minimize the total makespan of the grid environments. The algorithm uses genetic heuristic and searches the possible couples of the tasks and resources to find the best matching between them. The algorithm proposed in this paper is simulated and then compared with other well-known scheduling algorithms. The results obtained from comparing the proposed algorithm with Min-min, Max-min, Sufferage and RASA algorithms show that the proposed algorithm overcomes the weakness of the other ones by finding the best suitable task/resource pairs with lower makespan.

The remainder of this paper is organized as follows. Section 2 presents the related works introducing previously proposed scheduling algorithms. In section 3, background information about the grid environments and scheduling algorithms is presented and then the required preliminaries about the genetic algorithms are mentioned. Section 4 presents the genetic based scheduling algorithm and section 5 provides some results obtained from simulating various case studies within hypothesis grid environments. Also the comparison between the proposed algorithm and other related algorithms is provided in section 5. Finally, section 6 concludes the paper and presents future work.

2. Related Works

Due to some specific attributes of the grid environments such as relatively high communication costs between resources, most of the previously introduced scheduling algorithms are not applicable in these systems [4, 11]. Therefore, there have been ongoing attempts to propose new scheduling algorithms especially within heterogeneous distributed systems and grid environments [4, 7, 9, 11, 12, 13, 14, 15, 16, 17]. Some of these works are discussed below briefly.

He et al. [4] have presented a new algorithm based on the conventional Min-min algorithm. The proposed algorithm called quality of service (QoS) guided Min-min schedules tasks requiring high bandwidth before the others. Therefore, if the bandwidth required by different tasks varies highly, the QoS guided Min-min algorithm provides better results than

the conventional Min-min algorithm. Whenever the bandwidth requirement of all of the tasks is almost the same, the QoS guided Min-min algorithm acts similar to the Min-min algorithm.

Etminani et al. [10] have proposed a new algorithm using traditional scheduling algorithms, Max-min and Min-min. The proposed algorithm tries to select one of these two algorithms dependent on the standard deviation of the expected completion times of the tasks on each of the resources. Parsa et al. [11] have proposed a new task scheduling algorithm called RASA. RASA uses the advantages of both Min-min and Max-min algorithms and covers their disadvantages simultaneously. To achieve this, RASA firstly estimates the completion time of the tasks on each of the available resources, and then applies the Max-min and Min-min algorithms alternatively. Applying Max-min and Min-min methods alternatively, the Min-min strategy is used to execute small tasks before the large ones, and the Max-min strategy is used to avoid delays in the execution of large tasks and to support concurrency in the execution of large and small tasks. Experimental results reported in [11] show that RASA demonstrates relatively lower makespan in comparison with both Min-min and Max-min algorithms within grid environments.

Khanli et al. [16, 17] have presented QoS based scheduling solutions in a specific architecture called Grid-JQA. This scheduling solution applies an aggregation formula that is a combination of parameters together with weighting factors to evaluate QoS. Despite outperforming the Max-min, Min-min, and Sufferage, the algorithm proposed in [17] is not practical and seems to be an unpractical mathematical solution. Munir et al. [13] have presented a task scheduling algorithm for grid environments called QoS Sufferage. This algorithm considers network bandwidth and schedules tasks based on their bandwidth requirements as the QoS guided Min-min algorithm does. Compared with the Max-min, Min-min, QoS guided Min-min and QoS priority grouping algorithms, QoS Sufferage obtains smaller makespan.

Wang et al. [12] have presented a genetic-algorithm-based approach to dispatch and schedule subtasks within grid environments. Subtasks are produced from decomposition of tasks in grid manager and they should be scheduled appropriately to reach minimum completion time for the related entire task. The genetic-algorithm-based approach separates the matching and scheduling representations and provides independence between the chromosome structure and the details of the communication subsystem. Furthermore, the algorithm considers the overlap existing among all computations and communications that obey subtask precedence constraints. The simulation tests presented in [12] for small-sized problems (e.g., a small number of subtasks and a small number of machines), shows that the genetic-algorithm-based approach can found the optimal solution for these types of problems. The results obtained from simulation of the large size problems showed that this approach outperformed two non-evolutionary heuristics and a random search. Levitin et al. [15] have proposed a genetic algorithm to distribute execution blocks within grid resources. The aim of the algorithm is to find the best match between execution blocks and grid resources in which the reliability and/or expected performance of the task execution in the grid environment maximizes. The illustrative and numerical examples presented in [15] show that the proposed algorithm can find suitable solution for the problem.

3. Background Information

In this section, the required information about the grid computing environments, the scheduling algorithms and genetic optimization approach are presented. This information is used in the next section.

3.1. Grid Computing Environments

Grid computing is a newly developed technology for large scale systems to use the computational capabilities of the geographically distributed resources. The grid concepts and technologies have been introduced by Foster and Kesselman in 1998 [6]. Albeit before formally introducing the grid concepts, many efforts had been done to coordinate distributed resources in wide area networks. But the grid has introduced the new concepts in allocating and managing the distributed resources and focusing on central components. The initial definition of the grid provided by Foster and Kesselman in 1998 is as follows [6]:

- "A computational grid is a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities."

Since this early definition of the grid, there have been a number of other attempts to define the grid environments. For example the following are two other definitions [18, 19]:

- "A grid is a software framework providing layers of services to access and manage distributed hardware and software resources", or
- "A widely distributed network of high-performance computers, stored data, instruments, and collaboration environments shared across institutional boundaries".

The joint point of all the presented definitions is coordinated resource sharing and problem solving in dynamic and multi institutional virtual organizations [6, 15, 17, 18, 19]. The main goal of the grid computing is to provide the users the ability to harness the computational power of the large numbers of heterogeneous resources distributed within grid environment. To achieve this, the resource management system (RMS) or grid manager (GM) [11, 15, 19, 20] receives users' tasks and then schedules them to the available resources. To do this, a target factor should be defined for task scheduling and the scheduling algorithm should be performed considering the defined factor. After determining a suitable factor, an appropriate resource to execute a specific task is determined and then the task is sent to the resource to be executed. When the resource finishes the execution of the assigned task, it delivers the results to the user via the RMS. Finally, the RMS integrates the received results into the entire task output to respond to the user.

Resources existing in the grid environment are connected to each other via communication links. A communication link provides resources to converse and exchange data and results with the RMS. Several link topology models covering the spectrum from centralized models to fine-grained fully distributed models can be envisaged; among them single node, star, ring, tree, graph, and hybrid models are popular [19]. In this paper a very simple and straightforward link topology, star, is considered. In this topology, the RMS is the root of the tree and other connected nodes are the computational resources which are members of the grid. The star topology presents an abstract view of the grid resources and it is one of popular topologies used in most research works. Based on this topology, only the RMS can send the task data to the grid resources and since there is no communication links between the resources, they cannot have a direct communication with each other. Therefore, task scheduling algorithm should be run on the RMS and all of the tasks should be delivered by the RMS. The RMS is responsible for receiving the tasks, scheduling them and delivering the results to the grid users.

3.2. Task Scheduling Algorithms

Suppose that m resources R_j (j = 1, ..., m) have to process n tasks T_i (i = 1, ..., n). A schedule for each task is an allocation of one or more time intervals to one or more resources. The expected execution time E_{ij} of task T_i on resource R_j is defined as the amount of time taken by R_j to execute T_i where R_j has no load when T_i is assigned. The expected completion time C_{ij} of task T_i on resource R_j is defined as the wall-clock time on which R_j completes T_i (after having finished any previously assigned tasks). Let b_i denote to the beginning of the execution of task T_i. From the above definitions, the expected completion time C_{ij} can be written as Eq. 1.

$$C_{ij} = b_i + E_{ij} \quad (1)$$

Let C_i be the completion time for task T_i, and it is equal to C_{ij} where resource R_j is assigned to execute task T_i. The makespan for the complete schedule is then defined as Eq. 2.

$$Max_{T_i \in K}(C_i) \quad (2)$$

Where K is the set of tasks which has been assigned to the resources to be executed. Makespan is a measure of the throughput of the heterogeneous computing systems (e.g. computational grids) [7, 8, 11].

Lots of scheduling algorithms are proposed to assign the tasks to the resources by considering one or more QoS parameters. These algorithms show different performances based on the environment in which they are applied. The traditional parallel scheduling problem is to schedule the set of subtasks belonging to an application on the parallel machines to reduce the turnaround time. In a grid environment, the scheduling problem can be defined as scheduling of a set of tasks from different users on a set of computing resources to minimize the completion time of a specific task or the makespan of a system. Also, other parameters such as load balancing, task's execution time, system throughput, service reliability, service cost, system utilization, and so forth can be considered as goal of scheduling [10, 11, 19, 20].

Generally, the scheduling algorithms are divided into two basic categories; immediate mode scheduling and batch mode scheduling [8, 11]. In the immediate mode, a task is mapped onto a resource as soon as it arrives at the scheduler. In the batch mode, tasks are not mapped onto the resources as they arrive; instead they are collected into a set that is examined for mapping at prescheduled times called mapping events. The independent set of tasks which is considered for mapping at the mapping events is called a meta-task. Some algorithms estimate the execution time of the tasks on the resources, and then assign each task to the resource with the minimum expected execution time. These algorithms are named minimum execution time (MET) algorithms. The minimum completion time (MCT) algorithms assign each task to the resource which results in that task's shortest completion time [4, 7, 8, 11]. This causes some tasks to be assigned to the resources that do not have the minimum execution time for them.

There are several task scheduling algorithms attempting to minimize the makespan or overall completion time of all of the tasks in distributed systems. These algorithms are used as benchmarks to evaluate the new proposed algorithms. Min-min and Max-min algorithms schedule tasks considering the completion time of the tasks on the resources. Min-min heuristic is one of the batch mode scheduling schemes used as a benchmark for many batch mode mappings [7, 8]. The Min-min algorithm can be applied to wide range of systems such as multiprocessors and homogenous and heterogeneous distributed systems by

applying trivial modifications. This algorithm shows relatively low makespan in comparison with other similar algorithms. The Min-min algorithm begins with the set *U* of all unscheduled tasks. Then, the set of minimum completion times for each of the tasks existing in *U* is found. Next, the task with the overall minimum completion time existing in unscheduled tasks is selected and then assigned to the corresponding resource (hence it is named Min-min). Last, the newly scheduled task is removed from *U*, and the process repeats until all tasks are scheduled [8, 11]. The Max-min heuristic is similar to the Min-min heuristic and has the same complexity as the Min-min heuristic. It differs from the Min-min algorithm in that once the resource that provides the shortest completion time is found for every task; the task that has the maximum shortest completion time is determined and then assigned to the corresponding resource. The Max-min algorithm is likely to do better than the Min-min heuristic in the cases where there are many shorter tasks than longer ones.

RASA is a new proposed algorithm which combines two conventional algorithms, Min-min and Max-min, and achieves lower makespan in comparison with the basic algorithms [11]. RASA uses the number of the available resources when it determines the resource which should be allocated to a specific task. Whereas the number of the available resources is odd, the Max-min method is applied to the unscheduled tasks and when the number of the resources is even, the Min-min strategy is used. With this mechanism, RASA gives low makespan and consequently high throughput. Sufferage algorithm is a deterministic algorithm to schedule tasks within resources to achieve low makespan [7, 14]. The Sufferage heuristic is based on the idea that better mappings can be generated by assigning a resource to a task that would *suffer* most in terms of expected completion time if that particular resource is not assigned to it. In some of the simulated situations, Sufferage shows lower makespan compared to Min-min, Max-min and RASA algorithms. In this paper, four above mentioned algorithms are implemented and compared to the proposed genetic based algorithm.

3.3. Genetic Optimization Approach

Genetic algorithms (GAs) [12, 15, 21, 22] are a family of computational models inspired by evolution. These algorithms encode a potential solution to a specific problem on a simple chromosome-like data structure and apply recombination operators to these structures. GAs are often viewed as function optimizers, although the range of problems to which GAs have been applied is quite broad [21]. Since the GA is a type of searching algorithms, it searches a solution space for an optimal solution to a problem. The key characteristic of the GA is how the searching is done. Actually, the algorithm creates a *population* of possible solutions to the problem and lets them *evolve* over multiple generations to find better and better solutions [22]. The generic form of the genetic algorithm is shown in Figure 1.

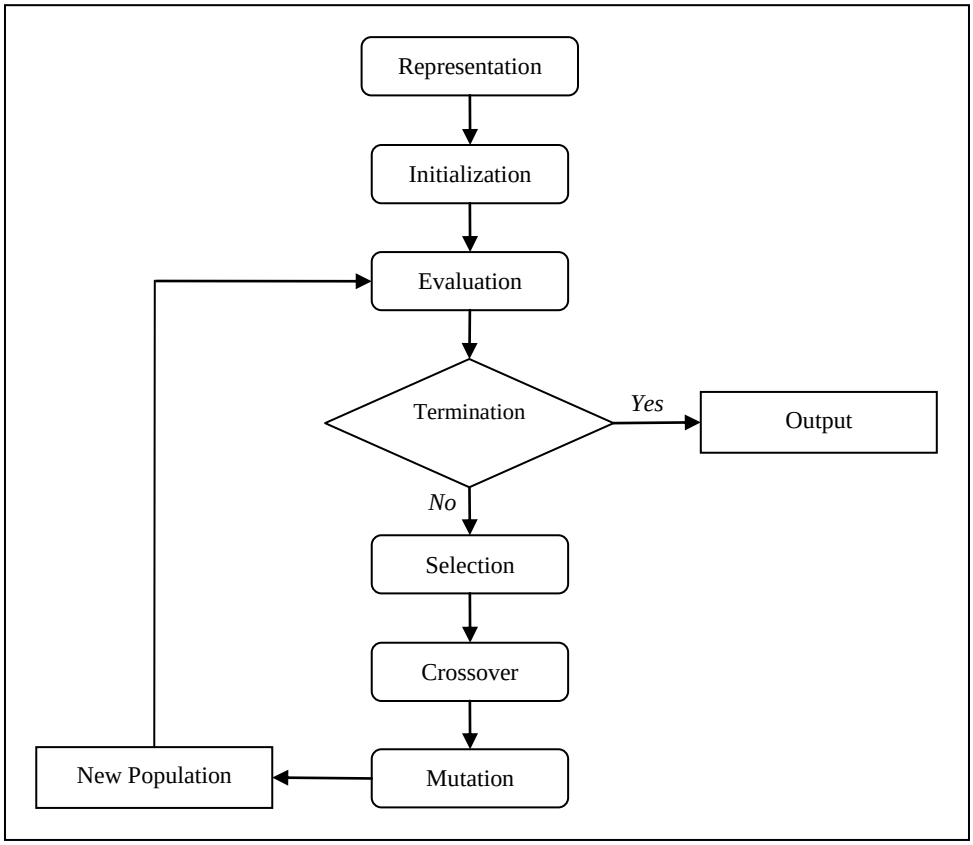

Figure 1. Steps of Genetic Algorithms

In the first step, it is necessary to encode any possible solution of the problem as a set of strings named as chromosomes. Each chromosome represents one solution to the problem, and a set of chromosomes is referred to as a population. The next step is to generate an initial population from valid chromosomes. A random set of chromosomes is often used as initial population. This initial population is the first generation from which the evolution starts. After generating the population, an evaluation is done on the first generation and reproductive opportunities is allocated in such a way that those chromosomes which represent a better solution to the target problem are given more chances to reproduce than those chromosomes which are poorer solutions. The goodness of a solution is typically defined with respect to the current population.

During each successive generation, a proportion of the existing population is selected to breed a new generation. In this step, certain selection methods such as roulette wheel selection, rank selection, tournament selection and so forth can be used to generate the new population. There are two key concepts during the selection of the parents, elitism and diversity. Selection algorithms should reserve these two concepts to reach a near optimal (not local optimal) solution for the problem in an acceptable time. The cross-over occurs by mixing the two solutions together to produce two new individuals. During each generation, there is a small possibility for each individual to mutate, which will change the individual in some small way. After crossover and mutation, the new population is generated and evaluation can be applied to the new

chromosomes. Crossover and mutation procedures should preserve validity of the newly obtained strings given where parent strings are valid.

The loop of chromosome generations is terminated when certain conditions are met. When the termination criteria are met, the elite chromosome as the best solution and its corresponding objective function value as the optimized value are returned.

4. The Proposed Algorithm

The task scheduling algorithm proposed in this paper is based on GA. To propose a new GA, the steps specified in Figure 1 should be addressed appropriately. Therefore, in the follow, the phases of the algorithm are presented step by step.

To represent the scheduling problem, the strings of integer numbers are used. Assume there are n independent tasks and m available resources to execute these tasks. Let first n integers (1, 2, …, n) in a string representing the scheduling problem denote the tasks whereas the numbers from $n + 1$ to $n + m$ represent available resources (any number $n + i$ corresponds to resource i). The greatest number ($n + m$) always takes the last position in a code string. The rule of representation decoding is as follows:

Any sequence of adjacent task numbers executed by the nearest resource placed in the right hand of the tasks.

Consider several examples for $n = 5$ and $m = 3$, that are decoded and interpreted in Table 1. The numbers corresponding to the resources are marked in bold.

Table 1. Examples of Task/Resource Encoding

String code	Interpretation
12**6**37458	Tasks 1 and 2 are assigned to resource 1, task 3 is assigned to resource 2 and tasks 4 and 5 are assigned to resource 3.
123**78**456	Tasks 1, 2 and 3 are assigned to resource 2, tasks 4 and 5 are assigned to resource 1 and no task is assigned to resource 3.
72318546	Tasks 2, 3 and 1 are assigned to resource 3, tasks 5 and 4 are assigned to resource 1 and no task is assigned to resource 2.
16273484	An invalid string code.

As it is mentioned before, a random set of chromosomes is often used as the initial population. In the proposed algorithm, the first generation composed of 30 initial individuals from valid chromosomes. These chromosomes are generated randomly. After that, each of the chromosomes generated in the previous step is evaluated. Each chromosome is associated with a fitness value which is, in our algorithm, the completion time of the tasks represented by this chromosome. In other words, a fitness value is assigned to each of the chromosomes and the objective of the GA is to find a chromosome with optimal fitness value. In the algorithm proposed in this paper, each chromosome with a smaller fitness value represents a better solution.

After calculating the fitness value for each of the chromosomes, a proportion of the existing population should be selected to produce a new generation. To do this, we use two well-known selection methods, tournament and rank selections, to select parents for

offspring. In the crossover step, all of the chromosomes selected in the selection phase are paired up, and with a crossover probability they are crossed over. To use crossover probability, first a random number *r* in range [0, 1] is generated and then if the number is less than crossover probability, two pairs are crossed over else they are copied as two new chromosomes to the new population. The crossover procedure used in the proposed algorithm acts as follows: for each couple of chromosomes, two cross over points are randomly selected between 1 to $n + m$. Code string from beginning of the chromosome to the first crossover point is copied from one parent (assume parent 1), the order of the genes existing in the part from the first to the second crossover point are inherited from second parent (assume parent 2) and the rest is copied from the first parent (parent 1). The following is an example of the crossover procedure on code strings described in Table 1. In this example, two random numbers are 2 and 7.

Parent 1: 12**6**3**7**45**8**

Parent 2: **7**231**8**54**6**

Children: 12**7**34**6**5**8**

After the crossover, for each of the genes of the chromosomes, the gene will be mutated to any one of the codes with a mutation probability. The mutation process transforms a valid chromosome into another valid one that may or may not already be in the current population. To perform mutation process on the chromosomes obtained from previous phase, a random number *r* in range [0, 1] is generated. If number *r* is less than the mutation probability, the selected chromosome is mutated else it is copied as a new chromosome to the new population. The mutation method used in our algorithm just swaps genes initially located in two randomly chosen positions of a chromosome selected for mutation. Both crossover and mutation methods introduced above are acceptable in our case and can be used to generate a new population.

As mentioned before, a termination condition should be considered to terminate the iteration and report the result. In our algorithm, the number of generations is used to specify the termination condition, but in general, other criteria such as allocation constraints (e.g. time), manual inspection, combination of the conditions and so forth can be used, too.

5. Simulation Results

To measure the performance of the proposed algorithm and compare the makespan of the algorithm to the others, randomly generated scenarios are considered. As it is mentioned in subsection 3.2, the makespan of the proposed algorithm is compared with the makespan of four well-known scheduling algorithms, Min-min, Max-min, RASA and Sufferage. In our simulations, various case studies are considered and the above mentioned algorithms are compared to each other. In almost all of the simulations, the number of the iterations in termination condition, number of the initial population, size of the selection set and crossover and mutation probabilities are set to 1000, 30, 20, 0.9 and 0.2, respectively. In the first simulation, the number of the resources is assumed to be 2 and the numbers of the tasks vary from 5 to 50. Figure 2 shows the makespan of the above mentioned algorithms in the first experiment.

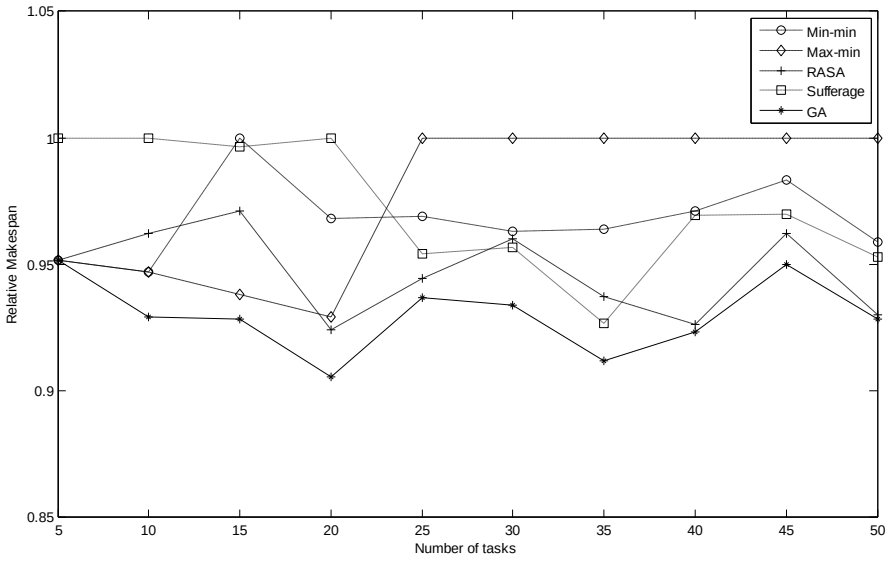

Figure 2. The Relative Makespan of the Five Scheduling Algorithms, Min-min, Max-min, RASA, Sufferage and GA in the First Experiment.

As shown in Figure 2, in almost all of the scenarios, the makespan of the GA is lower than other algorithms' makespans. Figure 3 shows the makespan of the algorithms in the second experiment in which the number of the resources is equal to 10 and the numbers of the tasks vary from 10 to 50. The results obtained from the second experiment more emphasis the first experiment's results. Indeed, it can be mentioned the proposed algorithm shows lower makespan for different numbers of resources and tasks (light and heavy loads).

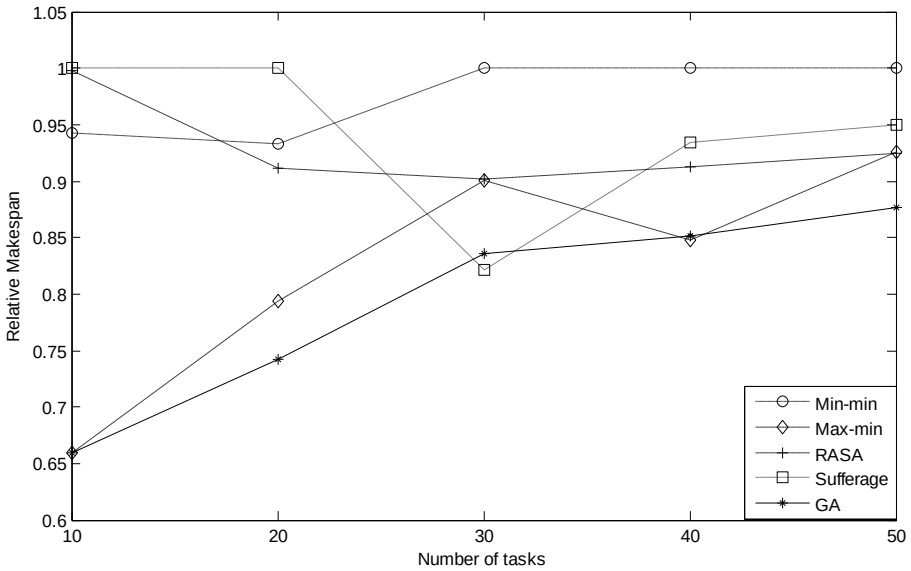

Figure 3. The Relative Makespan of the Five Scheduling Algorithms, Min-min, Max-min, RASA, Sufferage and GA in the Second Experiment.

It should be noticed that the makespan of the Min-min and Max-min algorithms is very depend on the number of small and large tasks, and therefore; in the random generated scenarios in which the size of the tasks is randomly varied, the makespan of these algorithms did not follow a specific pattern. Also, RASA and Sufferage algorithms show low makespan in comparison with Min-min and Max-min algorithms, but the makespan of the GA is lower than them.

Figure 4 shows the makespan of the five mentioned algorithms in eight different case studies. In the first four case studies, the proportion of the tasks to the resources is relatively low number and in the four remaining cases, this proportion is relatively large number. These two cases are considered to simulate light and heavy load in grid environments, respectively.

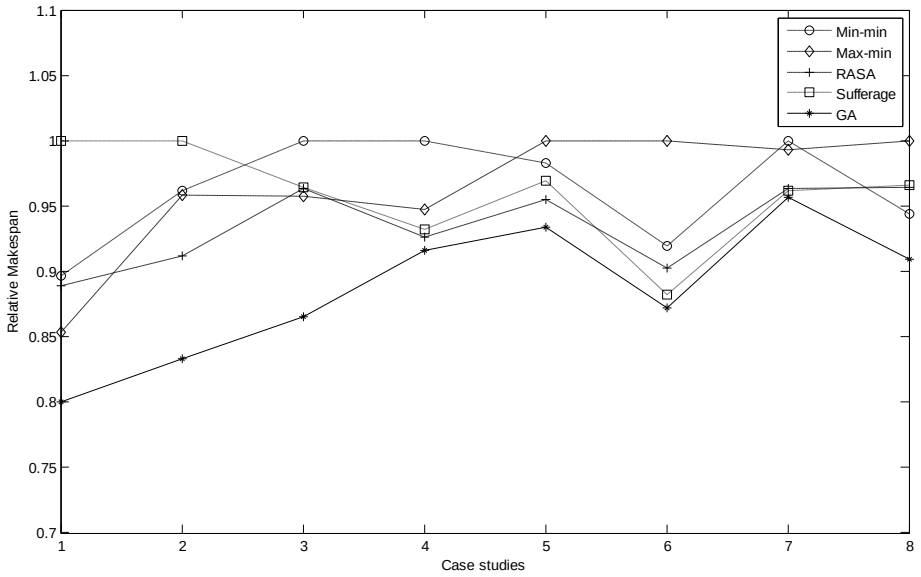

Figure 4. The Relative Makespan of the Five Mentioned Scheduling Algorithms where Cases 1-4 and 5-8 Show the Grid Environments with Light and Heavy Loads, Respectively.

In all of the above mentioned scenarios, the number of the iterations is used to determine the termination condition of the algorithm. This number is set to 1000 and lowers in the above experiments. Based on the scenarios studied in this paper, the iteration number equal to 1000 is an acceptable value in almost all of the cases. In this point of view, it can be mentioned that the GA proposed in this paper is early convergence. Figure 5 demonstrates the convergence of the GA in case study 2 given in Figure 4. Also, Figure 6 shows the convergence of the proposed algorithm when the number of the tasks and resources are equal to 50 and 10, respectively. As Shown in Figures 4 and 5, the variations in the makespans of new populations after 500 iterations are very trivial and the best solution can be found after 500-1000 iterations.

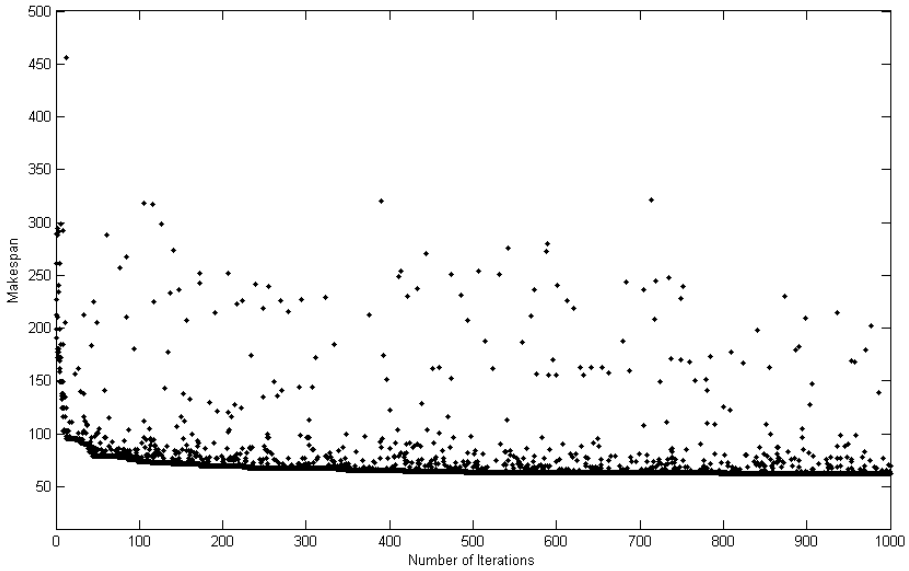

Figure 5. The Convergence of the Proposed Algorithm in Case Study 2 given in Figure 4.

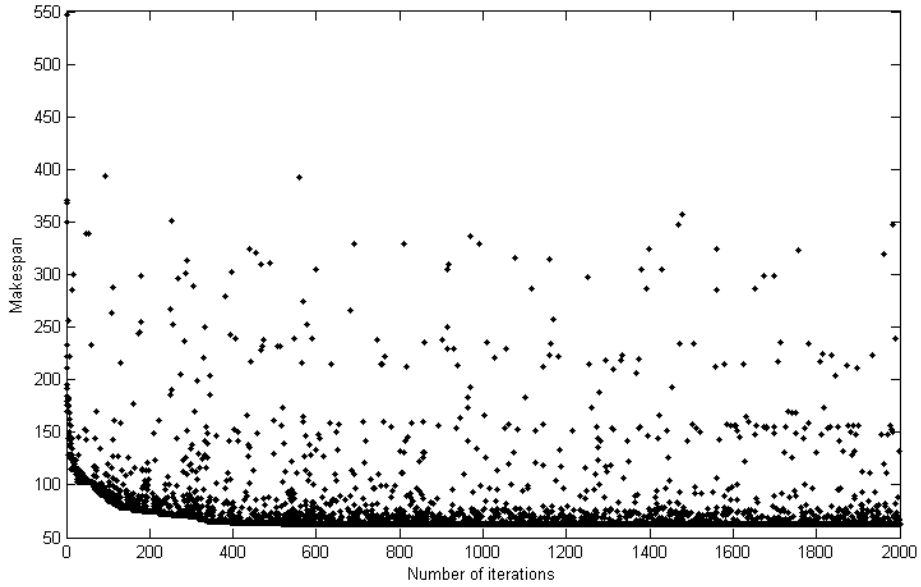

Figure 6. The Convergence of the Proposed Algorithm through 2000 Iterations where the Number of the Tasks and Resources are Equal to 50 and 10, Respectively.

6. Conclusions and Future Work

Task scheduling within grid environments is an NP-complete problem, so proposing new heuristic approaches to find the best possible matches between the grid tasks and resources is one of the interesting issues in grid environments. To fulfill this

requirement in computational grids, a new scheduling algorithm based on genetic approach is presented in this paper. The proposed algorithm can be applied to schedule tasks within grid environments to reduce the makespan of the tasks and thereby increase the throughput of the environment. The GA proposed in this paper not only uses very simple method to represent the resource allocation but also exposes lower makespan values in comparison to the well-known scheduling algorithms. Furthermore, the proposed GA converges to the suitable solution for a low number of iterations.

There are numbers of research issues remaining open for future works. One can use other QoS measures such as delay and finish times of the tasks, deadline and priority constraints of the tasks and so on to evaluate the chromosomes of the population. Taking into account other criteria (e.g. cost of scheduling, grid resources idle times, number of the waiting tasks to be processed within grid resources, fault tolerance issues, performance and reliability of the grid environment and so forth) may result in new scheduling algorithms in the grid environments. In addition, combining two or more measures to propose a general objective function is an interesting issue in this field of research.

References

[1] R.S. Montero, E. Huedo, I.M. Llorente, "Benchmarking of High Throughput Computing Applications on Grids", Journal of Parallel Computing, Vol. 32, 2006, pp. 267-279.

[2] Y. Xue, Y. Wang, J. Wang, Y. Luo, Y. Hu, S. Zhong, J. Tang, G. Cai, Y. Guan, "High Throughput Computing for Spatial Information Processing (HIT-SIP) System on Grid Platform", In: European Grid Conference 2005, Editors: G. Goos, J. Hartmanis, J.V. Leeuwen, LNCS, Vol. 3470, 2005, pp. 40-49.

[3] Condor Project, http://www.cs.wisc.edu/condor/overview/

[4] X. He, X.H. Sun, G.V. Laszewski, "QoS Guided Min-min Heuristic for Grid Task Scheduling", Journal of Computer Science and Technology, Vol. 18, 2003, pp. 442-451.

[5] H.C. Hsu, T.L. Chen, K.C. Li, "Performance Effective Pre-scheduled Strategy for Heterogeneous Grid Systems in the Master Slave Paradigm", Journal of Future Generation Computer Systems, Vol. 23, 2007, pp. 569-579.

[6] I. Foster, C. Kesselman, "The Grid: Blueprint for a New Computing Infrastructure", Morgan Kaufmann, 1999.

[7] M. Maheswaran, S. Ali, H.J. Siegel, D. Hensgen, R.F. Freund, "Dynamic Mapping of a Class of Independent Tasks onto Heterogeneous Computing Systems", Journal of Parallel and Distributed Computing, Vol. 59, 1999, pp. 107-131.

[8] T.D. Braun, H.J. Siegel, N. Beck, L.L. Boloni, M. Maheswaran, A.I. Reuther, J.P. Robertson, M.D. Theys, B. Yao, "A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems", Journal of Parallel and Distributed Computing, Vol. 61, 2001, pp. 810-837.

[9] L.Y. Tseng, Y.H. Chin, S.H. Wang, "A Minimized Makespan Scheduler with Multiple Factors for Grid Computing Systems", Journal of Expert Systems with Applications, Vol. 35, 2009, pp. 11118-11130.

[10] K. Etminani, M. Naghibzadeh, "A Min-min Max-min Selective Algorithm for Grid Task Scheduling", The Third IEEE/IFIP International Conference on Internet, Uzbekistan, 2007.

[11] S. Parsa, R. Entezari-Maleki, "RASA: A New Grid Task Scheduling Algorithm", International Journal of Digital Content Technology and its Applications, Vol. 3, 2009, pp. 91-99.

[12] L. Wang, H.J. Siegel, V.P. Roychowdhury, A.A. Maciejewski, "Task Matching and Scheduling in Heterogeneous Computing Environments Using a Genetic-Algorithm-Based Approach", Journal of Parallel and Distributed Computing, Vol. 47, 1997, pp. 1-15.

[13] E.U. Munir, J. Li, S. Shi, "QoS Sufferage Heuristic for Independent Task Scheduling in Grid", Information Technology Journal, Vol. 6, 2007, pp. 1166-1170.

[14] L.D. Briceno, M. Oltikar, H.J. Siegel, A.A. Maciejewski, "Study of an Iterative Technique to Minimize Completion Times of Non-Makespan Machines", The 21st International Parallel and Distributed Processing Symposium, California, 2007, pp. 1-14.

[15] G. Levitin, Y.S. Dai, "Optimal Service Task Partition and Distribution in Grid System with Star Topology", Reliability Engineering and System Safety, Vol. 93, 2008, pp. 152-159.

[16] L. Mohammad Khanli, M. Analoui, "Resource Scheduling in Desktop Grid by Grid-JQA", The 3rd International Conference on Grid and Pervasive Computing, 2008, pp.63-68.

[17] L. Mohammad Khanli, M. Analoui, "Grid_JQA: A QoS Guided Scheduling Algorithm for Grid Computing", The Sixth International Symposium on Parallel and Distributed Computing, 2007, pp. 242-249.

[18] M. Li, M. Baker, "The Grid Core Technologies", John Wiley & Sons Ltd., 2005.

[19] M.A. Azgomi and R. Entezari-Maleki, "Task Scheduling Modelling and Reliability Evaluation of Grid Services Using Coloured Petri Nets", Future Generation Computer Systems, Vol. 26, No. 8, 2010, pp. 1141-1150.

[20] S. Parsa and R. Entezari-Maleki, "Modeling and Throughput Analysis of Grid Task Scheduling Using Stochastic Petri Nets," The 2009 International Conference on Parallel and Distributed Processing Techniques and Applications, USA, 2009, pp. 458-464.

[21] D. Whitley, "A Genetic Algorithm Tutorial", Statistics and Computing, Vol. 4, No. 2, 1994, pp. 65-85.

[22] S.M. Thede, "An introduction to genetic algorithms", Journal of Computing Sciences in Colleges, Vol. 20, No. 1, 2004, pp. 115-123.

Authors

Reza Entezari-Maleki is currently a Ph.D. student in computer engineering (software) at the Department of Computer Engineering in Sharif University of Technology in Tehran, Iran. He received his B.S. and M.S. degrees in computer engineering (software) from Iran University of Science and Technology (IUST) in Tehran, Iran in 2007 and 2009, respectively. He is also a member of Iran's National Elites Foundation. His main research interests are grid computing, performance evaluation, performability and dependability modeling, and task scheduling algorithms.

Ali Movaghar received the B.S. degree in electrical engineering from the University of Tehran in 1977 and the M.S. and Ph.D. degrees in computer, information, and control engineering from the University of Michigan, Ann Arbor, in 1979 and 1985, respectively. He is currently a professor in the Department of Computer Engineering at Sharif University of Technology in Tehran, Iran, where he joined first as an assistant professor in 1993. He visited INRIA in France in 1984, worked at AT&T Laboratories from 1985-1986, and taught at the University of Michigan from 1987-1989. His main areas of interest are performance and dependability modeling, formal verification, wireless and mobile networks, and distributed real-time systems. He is also a senior member of the IEEE and a senior member of the ACM.