Cluster Comput (2012) 15:201–210
DOI 10.1007/s10586-011-0192-5

A new approach to the job scheduling problem in computational grids

Javad Akbari Torkestani

Received: 25 July 2011 / Accepted: 16 November 2011 / Published online: 10 December 2011

Abstract Job scheduling is one of the most challenging issues in Grid resource management that strongly affects the performance of the whole Grid environment. The major drawback of the existing Grid scheduling algorithms is that they are unable to adapt with the dynamicity of the resources and the network conditions. Furthermore, the network model that is used for resource information aggregation in most scheduling methods is centralized or semi-centralized. Therefore, these methods do not scale well as Grid size grows and do not perform well as the environmental conditions change with time. This paper proposes a learning automata-based job scheduling algorithm for Grids. In this method, the workload that is placed on each Grid node is proportional to its computational capacity and varies with time according to the Grid constraints. The performance of the proposed algorithm is evaluated through conducting several simulation experiments under different Grid scenarios. The obtained results are compared with those of several existing methods. Numerical results confirm the superiority of the proposed algorithm over the others in terms of makespan, flowtime, and load balancing.

Keywords Grid · Job scheduling problem · Resource allocation · Learning automata

1 Introduction

Grids are large scale collections of heterogeneous and autonomous systems from multiple administrative domains geographically distributed and interconnected by the wide area network. Grid implies to an extensive concept that is often referred to as the parallel system of the 1970s, the large-scale cluster system of the 1980s, and the distributed system of the 1990s. Grid technology is an emerging paradigm for large scale distributed computing. Scientific problems are becoming more and more complicated and hard to solve. These complex problems need a huge amount of computing resources that cannot be sufficiently provided in distributed or parallel systems. The computational Grid is a promising approach that exploits the synergy between a set of interconnected Grid nodes to reach a common goal to solve the massive computational problems [23, 25]. Grid resources can be freely added or withdrawn at any time according to the owners' discretion. Performance of the Grid nodes and their load frequently change with time. Grids allow the selection, aggregation, and sharing of the software and hardware resources of different computers in a distributed fashion. These systems provide pervasive, inexpensive, and reliable access to the high end computational capabilities. Although the Grid technology is still in the early stage of the research and development, due to the low cost of computing resources and recent advances in computing and wide-area networking, it has been extensively grown and moved from an obscure research subject to a practical, highly popular technology during the last decade [1–4].

Generally, the Grid job scheduling is defined as the process of decomposition of a large problem into a number of subtasks, and allocation of the subtasks to available computing resources. The efficiency of a Grid environment is strongly dependent on the job scheduling technique it follows. Different forms of the job scheduling problem are computationally hard to solve. It has been shown that the finding of an optimal solution to the job scheduling problem in heterogeneous Grid systems is known to be NP-hard [26] in general. Due to the hardness of the job scheduling

J. Akbari Torkestani (✉)
Young Researchers Club, Arak Branch, Islamic Azad University, Arak, Iran
e-mail: j-akbari@iau-arak.ac.ir

problem and the dynamicity and extensiveness of the Grid environments, there is an urgent need for an adaptive, efficient and cost effective algorithm to schedule the Grid jobs. A host of scheduling techniques have been already presented and implemented in different types of Grid [13, 24].

Due to the NP-hardness of the Grid job scheduling problem, the exact solutions can not be applied to the large problems that often arise under real Grid scenarios. Therefore, the approximation methods that suffice to find a near optimal solution are more promising approaches. Heuristics and meta-heuristics have shown to be useful approaches for solving a wide variety of hard-to-solve combinatorial and multi-objective optimization problems. Xhafa and Abraham [5] give a survey on computational models and heuristic methods for Grid scheduling problems. Population-based genetic algorithms (GA) [9–12, 18], particle swarm optimization (PSO) methods [7], ant colony optimization (ACO) techniques [14–16], Tabu Search [17, 18], simulated annealing (SA) algorithms [19, 20], Memetic Algorithms (MA) [21, 22] are several approaches that have been effectively used to solve the Grid job scheduling problem.

From the literature it is known that the existing job scheduling algorithms can not efficiently adapt to the dynamicity of the resources and environment conditions. Such scheduling methods make a single schedule for the entire workflow in advance, and then the tasks are conducted according to this schedule. This significantly degrades the Grid performance if the resource availability or the environmental conditions changes over time. In this paper, a learning automata (LA)-based algorithm is proposed to solve the job scheduling problem in Grid environments. In this method, two LA are associated with each scheduler. One is for scheduling the user submissions and another one for allocating the workload to the Grid computational resources. In the proposed learning automata-based job scheduling algorithm called LAJS, the workload that is assigned to each Grid node is proportional to its computational capacity, and each user client is permitted to submit its works whenever it needs. The proposed method is expected to shorten the execution time and to properly distribute the workload among the Grid resources. As the proposed algorithm proceeds, each user is assigned the portion of the Grid capacity as much as it needs. To show the performance of the proposed scheduling algorithm, several simulation experiments are conducted under several Grid scenarios. The results of the proposed algorithm are compared with those of FPSO [7] and GJS [10]. Simulation results show that the proposed algorithm outperforms the other methods in terms of makespan, flowtime, and load balancing.

The rest of the paper is organized as follows. Literature is reviewed in the next section. Section 3 gives an overview of the learning automata. In Sect. 4, a learning automata-based algorithm is proposed for job scheduling in Grid environments. In Sect. 5, the performance of the proposed algorithm is evaluated through simulation experiments, and Sect. 6 concludes the paper. List of acronyms used in this paper is provided in Table 2 (Appendix).

2 Related work

Martino and Mililotti [9] proposed genetic algorithm for finding a sub optimal solution to the job scheduling problem in Grid environments. In [10], Gao et al. presented two genetic programming methods for scheduling the jobs in a computational Grid, a single service method and a multiple service method. The former one estimates the completion time of a job when Grid provides only one type of service, and the latter one predicts the completion time of a job in a multiple service Grid. The proposed methods are developed at both system and application levels. At application level, an adaptive genetic-based scheduling algorithm is used to optimally assign the jobs to the Grid nodes in such a way that the average completion time of all the jobs is minimized. Carretero and Xhafa [11] used genetic algorithms for job scheduling on computational Grids to optimize the makespan and the total flowtime. The aim of this work is to show the power of genetic algorithm in the design of effective schedulers to assign a large number of jobs originated from large scale applications to Grid resources.

De Mello et al. [12] presented an improved version of the Route load balancing algorithm so called RouteGA (Route with Genetic Algorithm support). RouteGA schedules the tasks of the parallel applications considering computer neighborhoods. Although the results show that RouteGA performs well in large environments, there are several cases where the neighbors have neither enough computational capacity nor communication capability. In such cases, Route algorithm transmigrates the tasks until they become stabled in a Grid area having enough resources. The migration technique is very time consuming, and so reduces the Grid performance. To shorten the stabilization time, RouteGA [12] records the historical information of the parallel application behavior, computer capacities and load. Then, RouteGA uses this information to setting the parameters of a genetic algorithm that is responsible for optimizing the task allocation. Xhafa et al. [22] proposed two implementations of cellular MAs for job scheduling in Grid systems when both makespan and flowtime are simultaneously minimized. MA is a relatively new class of population-based heuristic methods in which the concepts of genetic algorithm (evolutionary search) and local search are combined [5]. Contrary to genetic and memteic algorithms that use unstructured population, cellular memetic algorithms support a structured population. The authors argue that the proposed algorithms are able to better control the tradeoff between the exploitation and exploration of the search space.

Bandieramonte et al. [14] proposed an ant colony-based job scheduling algorithm for computational Grids. The proposed algorithm is based on a different interpretation of pheromone trails. Chang et al. [15] designed a balanced ACO-based (BACO) job scheduling algorithm to appropriately assign the jobs to the Grid resources. BACO takes into consideration the resource status and job size to find the optimal scheduling. In this method, each job is associated with an ant, and ants are responsible for finding the available resources. To balance the load on the Grid resources, the status of the selected resource is modified by the local pheromone update function after job assignment, and the status of each resource for all jobs is modified by global pheromone update function after the completion of a job. In [16], Kant et al. presented an ant-based framework for job scheduling in Grid environments. Combining the ACO theory and dynamic information technique, this method aims at optimizing the job scheduling strategy and improving the resource utilization rate.

Cheng et al. [6] proposed the idea of applying the fuzzy set to improve the security and fault tolerance of the job scheduling process in Grid environments under failure-prone and risky conditions. They presented a fuzzy logic-based self adaptive replication scheduling algorithm to decide on the replication number of a user job. The authors claim that their method improves the performance of the Grid in terms of makespan, Grid utilization, and average waiting time. In [7], Liu et al. came up with the idea of combining the fuzzy logic and particle swarm optimization to find a near optimal solution of the job scheduling problem in Grid environments. In this method, the fuzzy matrices are used to represent the position and velocity of the particles in PSO [8]. Extending the particle representation from the real vectors to fuzzy matrices provides an efficient and dynamic mapping between the job scheduling problem and the particle. By this, the proposed fuzzy-based PSO method is able to dynamically generate an optimal schedule by which the completion of the tasks within a minimum period of time and efficient resource utilization are guaranteed.

3 Learning automata theory

A learning automaton [27, 28] is an adaptive decision-making unit that improves its performance by learning how to choose the optimal action from a finite set of allowed actions through repeated interactions with a random environment. The action is chosen at random based on a probability distribution kept over the action-set and at each instant the given action is served as the input to the random environment. The environment responds the taken action in turn with a reinforcement signal. The action probability vector is updated based on the reinforcement feedback from the environment. The objective of a learning automaton is to find the optimal action from the action-set so that the average penalty received from the environment is minimized. LA have been found to be useful in systems where incomplete information about the environment exists. LA are also proved to perform well in complex, dynamic and random environments with a large amount of uncertainties.

The environment can be described by a triple $E = \{\alpha, \beta, c\}$, where $\alpha = \{\alpha_1, \alpha_2, \dots, \alpha_r\}$ represents the finite set of the inputs, $\beta = \{\beta_1, \beta_2, \dots, \beta_m\}$ denotes the set of the values that can be taken by the reinforcement signal, and $c = \{c_1, c_2, \dots, c_r\}$ denotes the set of the penalty probabilities, where the element c_i is associated with the given action α_i. If the penalty probabilities are constant, the random environment is said to be a stationary random environment, and if they vary with time, the environment is called a non stationary environment. The environments depending on the nature of the reinforcement signal $\underline{\beta}$ can be classified into P-model, Q-model and S-model. The environments in which the reinforcement signal can only take two binary values 0 and 1 are referred to as P-model environments. Another class of the environment allows a finite number of the values in the interval [0, 1] can be taken by the reinforcement signal. Such an environment is referred to as Q-model environment. In S-model environments, the reinforcement signal lies in the interval $[a, b]$.

LA can be classified into two main families [27]: fixed structure learning automata and variable structure learning automata. Variable structure learning automata are represented by a triple $\langle \underline{\beta}\underline{\alpha}, L\rangle$, where $\underline{\beta}$ is the set of inputs, $\underline{\alpha}$ is the set of actions, and L is learning algorithm. The learning algorithm is a recurrence relation which is used to modify the action probability vector. Let $\alpha_i(k) \in \underline{\alpha}$ and $\underline{p}(k)$ denote the action selected by learning automaton and the probability vector defined over the action set at instant k, respectively. Let a and b denote the reward and penalty parameters and determine the amount of increases and decreases of the action probabilities, respectively. Let r be the number of actions that can be taken by learning automaton. At each instant k, the action probability vector $\underline{p}(k)$ is updated by the linear learning algorithm given in (1), if the selected action $\alpha_i(k)$ is rewarded by the random environment, and it is updated as given in (2) if the taken action is penalized.

$$p_j(k+1) = \begin{cases} p_j(k) + a[1 - p_j(k)]; & \text{for } j = i \\ (1-a)p_j(k); & \text{otherwise} \end{cases} \tag{1}$$

$$p_j(k+1) = \begin{cases} (1-b)p_j(k); \ \text{for } j = i \\ \left(\frac{b}{r-1}\right) + (1-b)p_j(k); & \text{otherwise} \end{cases} \tag{2}$$

If $a = b$, the recurrence equations (1) and (2) are called linear reward-penalty (L_{R-P}) algorithm, if $a \gg b$ the given equations are called linear reward-ϵpenalty ($L_{R-\epsilon P}$), and finally if $b = 0$ they are called linear reward-Inaction (L_{R-I}). In L_{R-I}, the action probability vectors remain unchanged when the taken action is penalized by the environment.

3.1 Variable action-set learning automata

A variable action-set learning automaton (VLA) is an automaton in which the number of actions available at each instant changes with time. It has been shown in [28] that a learning automaton with a changing number of actions is absolutely expedient and also ϵ-optimal, when the reinforcement scheme is L_{R-I}. Such an automaton has a finite set of n actions, $\underline{\alpha} = \{\alpha_1, \alpha_2, \ldots, \alpha_r\}$. $A = \{A_1, A_2, \ldots, A_m\}$ denotes the set of action subsets and $A(k) \subseteq \alpha$ is the subset of all the actions can be chosen by the learning automaton, at each instant k. The selection of the particular action subsets is randomly made by an external agency according to the probability distribution $\Psi(k) = \{\Psi_1(k), \Psi_2(k), \ldots, \Psi_m(k)\}$ defined over the possible subsets of the actions, where

$\Psi_i(k) = prob[A(k) = A_i | A_i \in A, 1 = i = 2^n - 1]$.

Let

$\hat{p}_i(k) = prob[\alpha(k) = \alpha_i | A(k), \alpha_i \in A(k)]$

denotes the probability of choosing action α_i, conditioned on the event that the action subset $A(k)$ has already been selected and $\alpha_i \in A(k)$ too. The scaled probability $\hat{p}_i(k)$ is defined as

$$\hat{p}_i(k) = \frac{p_i(k)}{K(k)} \tag{3}$$

where $K(k) = \sum_{\alpha_i \in A(k)} p_i(k)$ is the sum of the probabilities of the actions in subset $A(k)$, and $p_i(k) = prob[\alpha(k) = \alpha_i]$.

The procedure of choosing an action and updating the action probabilities in a VLA can be described as follows. Let $A(k)$ be the action subset selected at instant n. Before choosing an action, the probabilities of all the actions in the selected subset are scaled as defined in (3). The automaton then randomly selects one of its possible actions according to the scaled action probability vector $\hat{p}(k)$. Depending on the response received from the environment, the learning automaton updates its scaled action probability vector. Note that the probability of the available actions is only updated. Finally, the probability vector of the actions of the chosen subset is rescaled as

$p_i(k+1) = \hat{p}_i(k+1) \cdot K(k)$,

for all $\alpha_i \in A(k)$. The absolute expediency and ε-optimality of the method described above have been proved in [28].

4 Job scheduling algorithm

In this section, an adaptive learning automata-based algorithm is proposed for finding a near optimal solution to the job scheduling problem in computational Grids. To provide the sufficient background for understanding the proposed job scheduling algorithm, some definitions and preliminaries are presented first in Sect. 4.1. Then, in Sect. 4.2, the new job scheduling algorithm is presented.

4.1 Job scheduling problem definition

A computational Grid $\mathbb{G}$ can be defined as a triple $\langle \underline{\mathcal{G}}, \underline{P}, \underline{C} \rangle$, where $\underline{\mathcal{G}} = \{G_i | 1 \le i \le n\}$ denotes the set of n heterogeneous candidates Grid nodes, machines or sites connected by a wide area network, $\underline{P} = \{p_i | \forall G_i \in \underline{\mathcal{G}}\}$ denotes the set of processors of the Grid nodes $\underline{\mathcal{G}}$ (p_i^j denotes the jth processor of Grid node G_i), and $\underline{C} = \{\underline{c_i^j} | \forall p_i^j \in \underline{P}\}$ denotes the capacities (e.g., processing power, memory size, network bandwidth, soft ware availabilities, and etc.) of the set of processors $\underline{P}$. $\underline{c_i^j}$ is a vector specifying the capacities of processor p_i^j. Let $\sum_{i=1}^n p_i$ denotes the total number of processors or the computational capacity of Grid $\mathbb{G}$.

A job J_i can be modeled as a tuple $\langle \underline{\mathcal{T}_i}, \underline{R_i} \rangle$, where $\underline{\mathcal{T}_i} = \{\tau_i^j | 1 \le j \le k\}$ denotes the set of tasks into which job J_i is subdivided, and $\underline{R_i} = \{\underline{r_i^j} | 1 \le j \le k\}$ denotes the set of requirements (e.g., processing or resource requirements). A job is considered as a set of indivisible tasks, each allocated to execute on a Grid node in a non-preemptive mode. Vector $\underline{r_i^j}$ specifies the requirement set of task τ_i^j. $|\underline{\mathcal{T}_i}|$ denotes the size (required number of processors) of job J_i, and $\sum_{j=1}^k \underline{r_i^j}$ denotes the cost of job J_i. An application is a piece of a work which is defined as a job set.

Let n be the number of computational Grid nodes and m denotes the number of jobs. The Grid job scheduling problem can be modeled as a triple $\langle \mathbb{G}, \underline{J}, \Lambda \rangle$, where $\mathbb{G}\langle \underline{\mathcal{G}}, \underline{P}, \underline{C} \rangle$ describes the Grid environment, $\underline{J} = \{J_i \langle \underline{\mathcal{T}_i}, \underline{R_i} \rangle | i = 1, 2, \ldots, m\}$ denotes the set of m independent user jobs, and $\Lambda : \underline{J} \to \underline{\mathcal{G}} \times \underline{\sigma}$ defines a mapping for each job $J_i \in \underline{J}$ specifying the set of Grid nodes that are associated with that job and schedule $\underline{\sigma} : \underline{\mathcal{T}_i} \times \underline{\mathcal{G}} \to \underline{P} \times \underline{T}$. Assume that Grid node $G_{i'}$ is assigned to task $\tau_i^j \in \underline{\mathcal{T}_i}$. Schedule $\sigma(\tau_i^j, G_{i'})$ specifies pair $\{p_{i'}^{j'}, t_{i'}^{j'}\}$ for all $1 \le j' \le k'$, where $p_{i'}^{j'} \in \underline{P}$ is the processor of Grid node $G_{i'}$ allocated to task $\tau_i^j \in \underline{\mathcal{T}_i}$, and $t_{i'}^{j'} \in \underline{T}$ denotes the time at which processor $p_{i'}^{j'}$ finishes task τ_i^j. $\max_{\forall \tau_i^j \in \underline{\mathcal{T}_i}} t_{i'}^{j'}$ denotes the completion time of job J_i, and $\max_{\forall p_{i'}^{j'} \in G_{i'}} t_{i'}^{j'}$ denotes the time at which Grid node $G_{i'}$ completes all its scheduled tasks. Let $\underline{\delta}$ be the set of all possible schedules of Grid job scheduling problem $\langle \mathbb{G}, \underline{J}, \Lambda \rangle$. Scheduling $\underline{\sigma}^* \in \underline{\delta}$ is the optimal solution of $\langle \mathbb{G}, \underline{J}, \Lambda \rangle$, if $\underline{\sigma}$ meets all the requirements of job set $\underline{J}$, and $\chi(\underline{\sigma}^*) = \mathrm{Min}_{\chi \underline{\sigma} \in \underline{\delta}} \{\chi(\underline{\sigma})\}$, where $\chi(\underline{\sigma})$ denotes the cost of scheduling $\underline{\sigma}$.

4.2 The proposed algorithm

Due to the dynamicity of the Grid environment, heterogeneity and autonomy of the Grid nodes, and large scale and complicated user jobs, the performance of the Grid systems severely degrades, if there not exist an efficient job scheduling method. Several job scheduling techniques have been proposed in literature, however, the existing methods are generally unable to adapt with the dynamicity of the resources and the network conditions. In this paper, we aim to propose an adaptive job scheduling algorithm to cope with the dynamicity of the Grid environment, diversity of the computational capacity of the processing nodes, and different and a priori unknown user workloads.

Let us assume that there exist m different users $\{U_1, U_2, \ldots, U_m\}$, n heterogeneous Grid nodes $\{G_1, G_2, \ldots, G_n\}$, and q schedulers $\{\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_q\}$. Every user submits its jobs to a scheduler. One or more users may submit their works to one or more schedulers. Each scheduler can be in connection with one or more Grid nodes. Therefore, there is a many-to-many connection between the schedulers and Grid nodes. For simplicity but without lose of generality, let us assume that each user submits only one job at each stage. Therefore, there exist m different jobs $\{J_1, J_2, \ldots, J_m\}$ that must be scheduled on n different Grid nodes. As mentioned earlier, each job J_i might be subdivided into several tasks τ_i^j, where $1 \leq j \leq k$. Furthermore, there might be multiple processors $p_{i'}^{j'}$ available at each Grid node $G_{i'}$, where $1 \leq j' \leq k'$. Processor $p_{i'}^{j'}$ is responsible for a non-preemptive execution of task τ_i^j.

The proposed Grid job scheduling algorithm is independently run at each scheduler $\mathcal{S}_s$. In this method, each scheduler $\mathcal{S}_s$ is equipped with two learning automata A_{sU} and A_{sG}, the former one is for scheduling the users to submit their jobs proportional to their different and unknown workloads and the latter one for optimal allocation of the user works to the processors depending on their computational capacities. In the rest of this section, we present the detailed description of the proposed algorithm running at scheduler $\mathcal{S}_s$ to show the role of these two learning automata.

4.2.1 *Scheduling the user submissions*

Obviously, different users have different workloads and so need different computational capacities. Allocation of the same capacity of the computational Grid resources to different user clients leads to an undesirable condition under which some users have serious resource insufficiency (that lengthens the completion time of the jobs) while some others only use a small portion of the allocated computational capacity. Moreover, the workload of a user client may vary over time. To cope with the time-variable and different users' workloads, a good job scheduler must assign each user the Grid resources proportional to its need. The proposed job scheduling algorithm uses learning automaton A_{sU} to optimally schedule the user clients connected to scheduler $\mathcal{S}_s$ to submit their works.

Before stating the performance of scheduler $\mathcal{S}_s$, we describe how to form the action-set of learning automaton A_{sU}. This automaton aims at scheduling the user clients to submit their jobs. Each scheduler may receive the jobs from one or more users. Let $\{U_1, U_2, \ldots, U_{N_u}\}$ be the set of users that must be scheduled by $\mathcal{S}_s$. The action-set of automaton A_{sU} has N_u actions, each for a user client. That is, the action-set of A_{sU} is defined as

$$\underline{\alpha}_{sU} = \{\alpha_{sU}^i | \forall i \in \{1, 2, \ldots, N_u\}\}.$$

Selection of action α_{sU}^i means that scheduler $\mathcal{S}_s$ permits user U_i to submits its work. Let $\underline{p}_{sU} = \{p_{sU}^i | \forall i \in \{1, 2, \ldots, N_u\}\}$ be the action probability vector of the learning automaton A_{sU}. Clearly, action α_{sU}^i is selected with probability p_{sU}^i. All actions are initially chosen with the same probability $1/N_u$. This implies that all user clients initially have the same chance to submit their jobs to the system.

$\mathcal{S}_s$ schedules the user clients using a polling technique. It polls the users one-by-one to give them an opportunity to submit their works. At each stage, automaton A_{sU} chooses one of its possible actions (say, action α_{sU}^i) according to its action probability vector at random. By the selection of action α_{sU}^i, user U_i is implicitly granted the permission to send its work. Scheduler $\mathcal{S}_s$ checks the selected user clients to see if it has a ready job to submit. If so, scheduler $\mathcal{S}_s$ let user U_i send its work, queues the received works in the order of their arrival times, and increases the choice probability of (or rewards) the selected action α_{sU}^i (i.e., p_{sU}^i) by (1). Otherwise (i.e., user U_i has no job to submit), scheduler $\mathcal{S}_s$ decreases the choice probability of (or penalizes) the selected action by (2). By this, the workload placed on different user clients can be effectively balanced. After updating the internal state of the learning automaton A_{sU}, scheduler $\mathcal{S}_s$ initiates another stage and repeats the same operations as it did in the previous stage. As the scheduling algorithm proceeds, scheduler polls each user client (more probably) only when it has a job to submit, and this results in allocation of the Grid resources to each user proportional to its need.

4.2.2 *Scheduling the Grid resources*

In a Grid system, resources are heterogeneous, autonomous and dynamic in nature. Due to the dynamicity of the Grid environments, the resource characteristics are temporal and vary over time. For example, the resource availability is severely affected by the dynamics of the network condition such as link failure, collision, congestion, and etc. Under such circumstances, finding an optimal job scheduling strategy becomes incredibly hard. In the proposed job scheduling algorithm, learning automaton A_{sG} is responsible for finding a near optimal solution to the problem of allocating the users' jobs to the Grid nodes according to their computational capacities.

Let $\{G_1, G_2, \dots, G_{N_g}\}$ denotes the set of Grid resources that are scheduled by $\mathcal{S}_s$. Each Grid resource $G_{i'}$ includes a set of computational resources $p_{i'}^{j'}$ (for $1 \leq j' \leq k'$) having different processing powers. Each processing element $p_{i'}^{j'}$ might be scheduled by several schedulers simultaneously. As mentioned earlier, a job J_i could be subdivided into several tasks τ_i^j, where $1 \leq j \leq k$. The aim of learning automaton A_{sG} is to find a scheduling strategy to assign a computational resource $p_{i'}^{j'}$ to each task τ_i^j in such a way that the average execution time of the submitted jobs is minimized. The action-set of automaton A_{sG} is defined as

$$\underline{\alpha}_{sG} = \{\alpha_{sG}^{i'j'} | \forall p_{i'}^{j'} \in G_{i'}\}.$$

Selection of action $\alpha_{sG}^{i'j'}$ means that the processing element $p_{i'}^{j'}$ is chosen to assign to task τ_i^j. Let $q_{i'}^{j'}$ be the queue in which processing element $p_{i'}^{j'}$ stores the assigned tasks. Queue length is defined as the number of tasks waiting for process. Automaton A_{sG} schedules each task τ_i^j as follows:

As mentioned before, a vector $\underline{r_i^j}$ is associated with each task τ_i^j specifying its requirements. On the other side, $\underline{c_{i'}^{j'}}$ is a vector specifying the capacities of processor $p_{i'}^{j'}$ too. Processor $p_{i'}^{j'}$ can be assigned to task τ_i^j, if $\underline{r_i^j} \leq \underline{c_{i'}^{j'}}$. Obviously, there may exist processing nodes for which condition $\underline{r_i^j} \leq \underline{c_{i'}^{j'}}$ is not met. In this case, when scheduling task τ_i^j the actions corresponding to these nodes must be removed from the action-set of automaton A_{sG} as described in Sect. 3.1 on VLA. This increases the convergence rate and convergence speed of the automaton. To do so, for each task τ_i^j, scheduler $\mathcal{S}_s$ updates action-set $\underline{\alpha}_{sG}$ as

$$\underline{\alpha}_{sG} \leftarrow \underline{\alpha}_{sG} - \{\alpha_{sG}^{i'j'} | \underline{c_{i'}^{j'}} \ngeq \underline{r_i^j}\}.$$

The remaining actions are candidates that can be assigned to task τ_i^j. Automaton A_{sG} then randomly chooses one of its possible actions based on its updated action probability vector. Let us assume that action $\alpha_{sG}^{i'j'}$ (corresponding to processing node $p_{i'}^{j'}$) is selected by the learning automaton. Scheduler $\mathcal{S}_s$ checks queue $q_{i'}^{j'}$ to see if its length is shorter than or equal to dynamic average length L_s. If so, scheduler increases the choice probability of the selected action $\alpha_{sG}^{i'j'}$ by (1). Otherwise, scheduler decreases it by (2). Regardless of rewarding or penalizing the selected action, scheduler assigns task τ_i^j to processing node $p_{i'}^{j'}$. This method places the load on each Grid node proportional to its computational capacity. At the end of each assignment, all removed actions must be enabled again as described in Sect. 3.1. For each scheduler $\mathcal{S}_s$, dynamic average length L_s is defined as

$$\sum_{i'=1}^{N_g} \sum_{j'=1}^{k} length(q_{i'}^{j'}).$$

That is, L_s represents the average length of the queues of the processing elements of all Grid nodes associated with scheduler $\mathcal{S}_s$, i.e., $\{G_1, G_2, \dots, G_{N_g}\}$. For each processing node $p_{i'}^{j'}$, $length(q_{i'}^{j'})$ is updated as soon as $\alpha_{sG}^{i'j'}$ is selected by the learning automaton. Automaton A_{sG} impartially distributes the submitted workloads on different Grid nodes, each proportional to its computational capacity.

5 Experimental results

To study the performance of the proposed Grid job scheduling algorithm, we have conducted several simulation experiments under three Grid sizes: A small scale Grid system including 16 Grid nodes and 128 processing elements, a medium size Grid environment comprising 32 Grid nodes and 256 processing elements, and finally a Large scale Grid with 128 Grid nodes and 1024 processing elements. (A real large scale Grid system might be composed of several thousands nodes.) In conducted experiments, it is assumed that all the jobs only require the computational resources. Therefore, the job requirement and Grid capacity vectors only include the computational features of the job set and Grid nodes. For each Grid size, it is assumed that all Grid nodes have the same number of processing elements. The computational capacity of the processing elements is generated by a Gaussian probability distribution function with mean 1000 MIPS (million instruction per second) and variance 150. The nominal bandwidth of the network connecting every two Grid nodes is assumed to be 100 Mbps. The number of users is fixed at 50, 100, and 400 for small scale, medium scale, and large scale Grids, respectively. The total number of jobs that are submitted by all the Grid users is fixed at 1000, 2000, and 8000 for small, medium, and large scale Grids, respectively. The execution time of the jobs generated by each user is normally distributed with mean 500 million instructions (MI) and variance 100 million. Each job is composed of a number of tasks. Jobs are subdivided into k tasks, where k is randomly and uniformly selected from set $\{1, 2, 3, 4\}$. At each user client, the generation rate of the new jobs is Poisson distributed with rate (mean) $\{5, 10, 15, 20\}$. This rate is evenly drawn from set $\{5, 10, 15, 20\}$ at random for each experiment. To improve the precision of the reported results, each experiment is independently repeated 50 times and the obtained results are averaged over these runs. Table 1 summarizes the simulation parameters.

To show the outperformance of the proposed job scheduling algorithm, the obtained results are compared with those of FPSO (a fuzzy logic-based particle swarm optimization algorithm for job scheduling problem proposed by Liu et al. [7]), and GJS (an adaptive genetic programming-based Grid job scheduling algorithm proposed by Gao et al. [10]). In simulation experiments, the efficiency of the proposed job

Table 1 Simulation experiment setup

Simulation parameter	Description	Value
Number of tasks per job (k)	Uniform distribution	$U[1, 2, 3, 4]$
Job generation rate	Poisson distribution	$P[5, 10, 15, 20]$
Total number of jobs	Small scale	1000
	Medium scale	2000
	Large scale	8000
Number of users	Small scale	50
	Medium scale	100
	Large scale	400
Nominal bandwidth		100 Mbps
Execution time	Normal distribution	$N(500, 100)$ MI
Processor computational capacity	Normal distribution	$N(1000, 150)$ MIPS
Number of Grid nodes	Small scale	16
	Medium scale	32
	Large scale	128
Number of processors	Small scale	128
	Medium scale	256
	Large scale	1024

scheduling method is compared with the above mentioned algorithms in terms of makespan, flowtime, and load balancing.

5.1 Makespan

This metric is defined as the maximum execution time of all submitted jobs. In other words, makespan is the completion time of the latest task. Minimization of this metric implies that no job takes a long time to execute. Makespan is computed as

$$\max_{\forall G_{i'} \in \underline{\mathcal{G}}} \left\{ \max_{\forall p_{i'}^{j'} \in G_{i'}} \{t_{i'}^{j'}\} \right\}$$

or as

$$\max_{\forall \underline{T_i} \in \underline{J}} \left\{ \max_{\forall \tau_i^j \in \underline{T_i}} \{t_{i'}^{j'}\} \right\}$$

in *ms*.

Figure 1 shows the average makespan of different algorithms under different Grid scales. As it can be seen, the proposed Grid job scheduling algorithm, LAJS, significantly outperforms FPSO and GJS specifically in large Grid scales. This is because LAJS dynamically schedules both the user submissions at a higher level and the Grid resources at a lower level. In LAJS, the Grid resources that are assigned to each user client are proportional to its workload. User submission rate changes over time and LAJS adaptively tunes the user portion. This avoids misspending the Grid resources. Furthermore, the load that is places on each Grid resource is directly proportional to its computational capacity. This avoids the grid resource being too busy or idle.

For small and medium scale Grids, the results of FPSO are very close to those of GJS. However, for large scale Grids, FPSO significantly surpasses GJS. Comparing the results shown in Fig. 1, it can be seen that the gap between the proposed scheduling algorithm and the other two methods (FPSO and GJS) becomes more significant as the system scale grows. Numerical results show that the makespan of GJS is nearly two times longer than that of the proposed scheduler. Overall, FPSO is ranked far below the proposed scheduler and GJS lags behind FPSO.

5.2 Flowtime

Flowtime is another Grid performance metric that is defined as the sum of the completion time of all the jobs. An optimal scheduling method is expected to minimize the flowtime. This metric refers to the total time that the allocated processors are occupied with the execution of the assigned jobs. Flowtime is calculated as

$$\sum_{\forall G_{i'} \in \underline{\mathcal{G}}} \left\{ \max_{\forall p_{i'}^{j'} \in G_{i'}} \{t_{i'}^{j'}\} \right\}$$

in *ms*.

This experiment is devoted to compare the average flowtime of the proposed job scheduling algorithm with that of FPSO and GJS under different Grid scale scenarios. The obtained results are depicted in Fig. 2. As shown in this figure, the obtained results confirm the superiority of the proposed

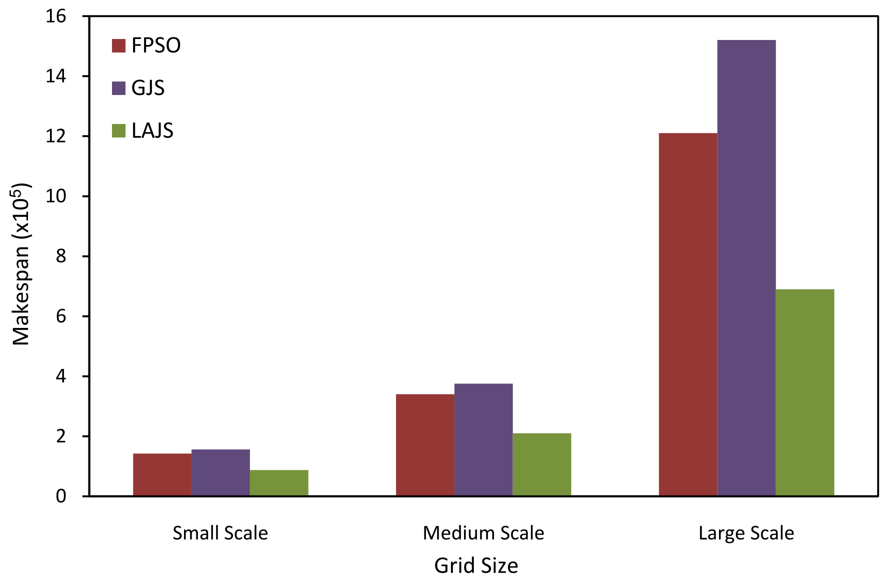

Fig. 1 Average makespan under different Grid scale

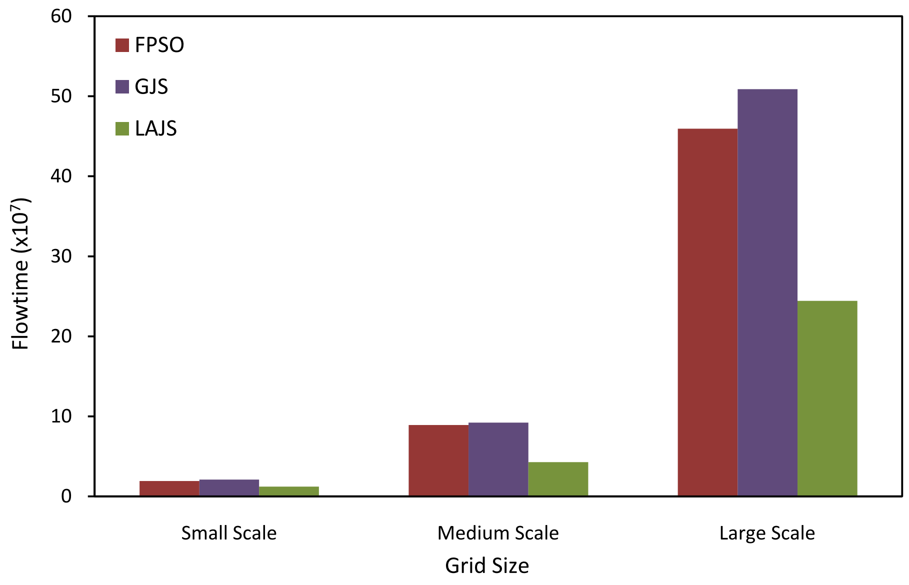

Fig. 2 Average flowtime under different Grid scale

algorithm over the other methods in terms of the flowtime, specifically in medium and large scale Grids. This could be due to the fact that LAJS is a time variable job scheduling technique in which the workload that is assigned to each Grid resource varies over time according to its available computational capacity. LAJS balances the arrival workloads on the Grid nodes. Contrary to the proposed algorithm, GJS and FPSO initially make a schedule for all the submitted jobs. Then, the tasks are executed according to the preplanned schedule. That is why GJS and FPSO fail under resource and network dynamics that LAJS can tolerate them.

As the number of submitted jobs increases, LAJS learns how to efficiently assign the jobs to the Grid nodes. That is, as the proposed algorithm proceeds, the learning automata associated with the scheduler converge to the optimal configurations by which the resource that is assigned to each user is proportional to its need and the workload that is placed on each Grid node is proportional to its capacity. This reduces the average completion time of the jobs, and so decreases the flowtime. From the results shown in Fig. 2, it can be seen that the flowtime of FPSO is slightly smaller than that of GJS in small and medium size Grids, and the gap between these two algorithms becomes more significant in large scale systems.

5.3 Load balancing

Figure 3 shows the load balancing of different job scheduling algorithms under different Grid scale scenarios. Load balancing represents the distribution of the workload allocated to the Grid nodes. A uniform workload distribution

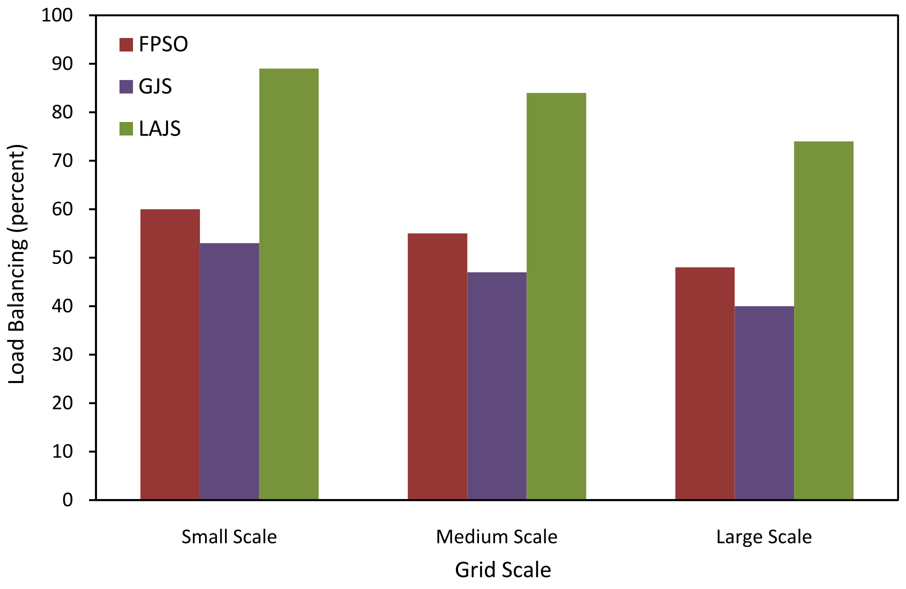

Fig. 3 Load balancing under different Grid scale

shows the load balancing only when all the Grid nodes have the same computational capacities. However, in case of different computational capacities, the workload which is submitted to each Grid node must be proportional to its capacity. In this case, the standard deviation of the completion time (i.e., the time at which a Grid node completes the execution of its last work) of the Grid nodes stands for the load balancing. This metric shows the difference between the completion times of different Grid nodes. Makespan and flowtime are minimized, if the workload placed on the Grid nodes is balanced. Load balancing increases as the standard deviation of the completion time decreases. Let $\mathbb{T}$ denotes the completion time of all the Grid nodes. Load balancing is computed as

$$\frac{\overline{\mathbb{T}} - \sigma_{\mathbb{T}}}{\overline{\mathbb{T}}} \times 100.$$

where $\overline{\mathbb{T}}$ and $\sigma_{\mathbb{T}}$ denote the mean and standard deviation of completion time $\mathbb{T}$, respectively.

From the results shown in Fig. 3, it can be seen that the proposed scheduling procedure distributes the submitted workloads among the Grid nodes considerably more balanced than FPSO and GJS for all Grid scales. This is due to the fact that LAJS takes into consideration the queue length of the processing elements to evenly distribute the workload within the Grid system. In LAJS, the queue length of each processing element (of a given scheduler $\mathcal{S}_s$) is adjusted by a dynamic threshold (i.e., L_s) in such a way that all processing elements have the queues of the same length and no lengthier than the dynamic average length L_s. Furthermore, workload allocation proportional to the computational capacity avoids very busy or idle processors and balances the workload within the Grid. From the results depicted in Fig. 3, it is observed that the load balancing slightly decreases as the Grid size grows. This can be due to the fact that making a balanced schedule becomes harder as the number of user submissions and Grid nodes increases. As expected, the schedules made by FPSO are more balanced than those made by GJS. That is why the flowtime and makespan of FPSO are shorter than those of GJS.

6 Conclusion

In this paper, a learning automata-based job scheduling algorithm was proposed for Grid environments. In the proposed algorithm, two learning automata are associated with each scheduler. The first automaton is responsible for scheduling the user submissions and the second one for optimal assignment of the tasks to the Grid resources in such a way that each user is assigned the portion of the Grid computational capacity proportional to its need. To show the efficiency of the proposed job scheduling algorithm, several simulation experiments were conducted for different Grid scales. The results of the proposed method were compared with those of a fuzzy logic-based particle swarm optimization job scheduling algorithm called FPSO and an adaptive genetic programming-based Grid job scheduling algorithm called GJS. Numerical results showed that the proposed algorithm significantly outperforms FPSO and GJS in terms of makespan, flowtime, and load balancing, specifically in large scale Grid environments. The obtained results also revealed that FPSO lags far behind the proposed scheduler and GJS is ranked below FPSO. Experiments showed that the makespan and flowtime of GJS are two times longer that those of LAJS.

Appendix

Table 2 List of acronyms

Acronym	Full term
ACO	Ant Colony Optimization
BACO	Balanced Ant Colony Optimization
FPSO	Fuzzy logic-based Particle Swarm Optimization
GA	Genetic Algorithm
GJS	Genetic programming-based grid Job Scheduling
LA	Learning Automata
LAJS	Learning Automata-based Job Scheduling
MA	Memetic Algorithm
PSO	Particle Swarm Optimization
RouteGA	Route with Genetic Algorithm support
SA	Simulated Annealing
VLA	Variable action-set Learning Automata

References

1. Tang, M., Lee, B.-S., Tang, X., Yeo, C.-K.: The impact of data replication on job scheduling performance in the Data Grid. Future Gener. Comput. Syst. **22**, 254–268 (2006)
2. Nakajima, Y., Sato, M., Aida, Y., Boku, T., Cappello, F.: Integrating computing resources on multiple Grid-enabled job scheduling systems through a Grid RPC system. J. Grid Comput. **6**(2), 141–157 (2008)
3. Tchernykh, A., Schwiegelshohn, U., Yahyapour, R., Kuzjurin, N.: On-line hierarchical job scheduling on Grids with admissible Allocation. J. Sched. **13**(5), 545–552 (2010)
4. Boyar, J., Favrholdt, L.M.: Scheduling jobs on Grid processors. Algorithmica **57**(4), 819–847 (2010)
5. Xhafa, F., Abraham, A.: Computational models and heuristic methods for Grid scheduling problems. Future Gener. Comput. Syst. **26**, 608–621 (2010)
6. Cheng, W., Congfeng, J., Xiaohu, L.: Fuzzy logic-based secure and fault tolerant job scheduling in Grid. Tsinghua Sci. Technol. **12**(S1), 45–50 (2007)
7. Liu, H., Abraham, A., Hassanien, A.E.: Scheduling jobs on computational Grids using a fuzzy particle swarm optimization algorithm. Future Gener. Comput. Syst. **26**, 1336–1343 (2010)
8. Liu, H., Abraham, A.: A hybrid fuzzy variable neighborhood particle swarm optimization algorithm for solving quadratic assignment problems. J. Univers. Comput. Sci. **13**(7), 1032–1054 (2007)
9. Di Martino, V., Mililotti, M.: Sub optimal scheduling in a Grid using genetic algorithms. Parallel Comput. **30**, 553–565 (2004)
10. Gao, Y., Rong, H., Zhexue Huang, J.: Adaptive Grid job scheduling with genetic algorithms. Future Gener. Comput. Syst. **21**, 151–161 (2005)
11. Carretero, J., Xhafa, F.: Using genetic algorithms for scheduling jobs in large scale Grid applications. J. Technol. Econ. Dev. **12**(1), 11–17 (2006)
12. de Mello, R.F., Andrade Filho, J.A., Senger, L.J., Yang, L.T.: Grid job scheduling using Route with genetic algorithm support. Telecommun. Syst. **38**(3–4), 147–160 (2008)
13. de Mello, R.F., Senger, L.J., Yang, L.T.: A routing load balancing policy for Grid computing environments. In: Proceedings of the 20th International Conference on Advanced Information Networking and Applications (AINA 2006), pp. 1–6 (2006)
14. Bandieramonte, M., Di Stefano, A., Morana, G.: An ACO inspired strategy to improve jobs scheduling in a Grid environment. In: Lecture Notes in Computer Science, vol. 5022, pp. 30–41 (2008)
15. Chang, R.-S., Changa, J.-S., Lina, P.-S.: An ant algorithm for balanced job scheduling in Grids. Future Gener. Comput. Syst. **25**, 20–27 (2009)
16. Kant, A., Sharma, A., Agarwal, S., Chandra, S.: An ACO approach to job scheduling in Grid environment. In: Lecture Notes in Computer Science, vol. 6466, pp. 286–295 (2010)
17. Xhafa, F., Carretero, J., Dorronsoro, B., Alba, E.: Tabu Search algorithm for scheduling independent jobs in computational Grids. Comput. Inform. J. **28**(2), 237–249 (2009)
18. Xhafa, F., Gonzalez, J.A., Dahal, K.P., Abraham, A.: A GA(TS) hybrid algorithm for scheduling in computational grids. In: Hybrid Artificial Intelligent Systems, Lecture Notes in Computer Science, vol. 5572, pp. 285–292 (2009)
19. Abraham, A., Buyya, R., Nath, B.: Nature's heuristics for scheduling jobs on computational Grids. In: Proceedings of the 8th IEEE International Conference on Advanced Computing and Communications, India (2000)
20. YarKhan, A., Dongarra, J.: Experiments with scheduling using simulated annealing in a Grid environment. In: Proceedings of GRID2002, pp. 232–242 (2002)
21. Xhafa, F.: A hybrid evolutionary heuristic for job scheduling in computational Grids. In: Studies in Computational Intelligence, vol. 75. Springer, Berlin (2007) (Chap. 10)
22. Xhafa, F., Alba, E., Dorronsoro, B., Duran, B.: Efficient batch job scheduling in Grids using cellular memetic algorithms. J. Math. Model. Algorithms **7**(2), 217–236 (2008)
23. Wu, J., Xu, X., Zhang, P., Liu, C.: A novel multi-agent reinforcement learning approach for job scheduling in Grid computing. Future Gener. Comput. Syst. **27**, 430–439 (2011)
24. Ramírez-Alcaraz, J.M., Tchernykh, A., Yahyapour, R., Schwiegelshohn, U., Quezada-Pina, A., González-García, J.L., Hirales-Carbajal, A.: Job allocation strategies with user run time estimates for online scheduling in hierarchical Grids. J. Grid Comput. **9**(1), 95–116 (2011). doi:10.1007/s10723-011-9179-y
25. Ghosh, P., Das, S.K.: Mobility-aware cost-efficient job scheduling for single-class Grid jobs in a generic mobile Grid architecture. Future Gener. Comput. Syst. **26**, 1356–1367 (2010)
26. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-completeness. Freeman, New York (1979)
27. Narendra, K.S., Thathachar, K.S.: Learning Automata: An Introduction. Prentice-Hall, New York (1989)
28. Thathachar, M.A.L., Harita, B.R.: Learning automata with changing number of actions. IEEE Trans. Syst. Man Cybern. **SMG17**, 1095–1100 (1987)

Javad Akbari Torkestani received the B.S. and M.S. degrees in Computer Engineering in Iran, in 2001 and 2004, respectively. He also received the Ph.D. degree in Computer Engineering from Science and Research University, Iran, in 2009. Currently, he is an assistant professor in Computer Engineering Department at Arak Azad University, Arak, Iran. Prior to the current position, he joined the faculty of the Computer Engineering Department at Arak Azad University as a lecturer. His research interests include wireless networks, multi-hop networks, fault tolerant systems, grid computing, learning systems, parallel algorithms, and soft computing.