Future Generation Computer Systems 27 (2011) 430–439

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

A novel multi-agent reinforcement learning approach for job scheduling in Grid computing

Jun Wu *, Xin Xu *, Pengcheng Zhang, Chunming Liu

Institute of Automation, College of Mechatronics and Automation, National University of Defense Technology, China

ARTICLE INFO

Article history:
Received 13 September 2009
Received in revised form
11 October 2010
Accepted 18 October 2010
Available online 19 November 2010

Keywords:
Multi-agent reinforcement learning
Load balancing
Job scheduling
Grid computing
Resource allocation
Coordination

ABSTRACT

Grid computing utilizes distributed heterogeneous resources to support large-scale or complicated computing tasks, and an appropriate resource scheduling algorithm is fundamentally important for the success of Grid applications. Due to the complex and dynamic properties of Grid environments, traditional model-based methods may result in poor scheduling performance in practice. Scalability and adaptability are among the key objectives of Grid job scheduling. In this paper, a novel multi-agent reinforcement learning method, called ordinal sharing learning (OSL) method, is proposed for job scheduling problems, especially, for realizing load balancing in Grids. The approach circumvents the scalability problem by using an ordinal distributed learning strategy, and realizes multi-agent coordination based on an information-sharing mechanism with limited communication. Simulation results show that the OSL method can achieve the goal of load balancing effectively, and its performance is even comparable to some centralized scheduling algorithm in most cases. The convergence property and adaptability of the proposed method are also illustrated.

1. Introduction

Multi-agent resource allocation is the process of distributing a number of items amongst a number of agents, and acts as a central matter of concern in both computer science and economics [1]. It is relevant to a wide range of application domains, such as network routing [2], public transportation [3] and Grid computing [4–6], where Grid computing is one of the most important applications of resource allocation or scheduling [7].

Grid computing enables the sharing, selection, and aggregation of geographically distributed heterogeneous resources and becomes an important solution paradigm for supporting complicated computing problems. However, there are still some technical challenges for Grids [5]. For a majority of Grid systems, the real and specific problem that underlies Grid computing is coordinated resource scheduling and problem solving in dynamic, multi-institutional virtual organizations, where an effective and efficient scheduling algorithm is fundamentally important [8,9]. Only with the help of a feasible scheduling policy, can the Grids speed up the task process and provide non-trivial services to users [10]. In the following, the job scheduling problem, which is the key issue for balancing the entire system load while completing all the jobs at hand as soon as possible, is studied (see Fig. 1).

In the past decade, there have been many advances in Grid job scheduling techniques. Various scheduling approaches, including model-based or model-free methods, either using centralized or decentralized mechanisms, have been developed for Grids. On the one hand, lots of algorithms have been studied for job scheduling problems in traditional parallel and distributed systems, such as FPLTF (Fastest Processor to Largest Task First), WQR (Work Queue with Replication) and FCFS (First Come First Serve) [11]. On the other hand, extensive research has been done for Grid scheduling problems, too. In traditional resource scheduling systems, such as Condor [12], PBS [13] and SGE [14], centralized schedulers work effectively since accurate and global information can be obtained. However, centralized or hierarchical resource allocation methods may suffer from the lack of scalability and fault-tolerance ability as well as having a single point of failure [15]. To overcome the scalability problem, some decentralized scheduling algorithms have been proposed. However, most existing decentralized schedulers, for example, in Condor-G [16] and AppleS [17], perform individual scheduling policies regardless of the other schedulers' decisions and may lead to serious synchronization problems in resource management. Finally, a Herd behavior will arise since schedulers run without central oversight and communication [18,19]. However, if job scheduling is carried out under the assumption of coordination, such as in Legion Federation [20] and Condor Flock P2P [21], the strong dependency on negotiation among schedulers and

* Corresponding address: College of Mechatronics and Automation, National University of Defense Technology, 410073 Changsha, Hunan, China. Tel.: +86 731 84574980; fax: +86 731 84573392.
E-mail addresses: xinxu@nudt.edu.cn, aresnudt@yahoo.com.cn, xuxin_mail@263.net (X. Xu).

0167-739X/$ – see front matter © 2011 Published by Elsevier B.V.
doi:10.1016/j.future.2010.10.009

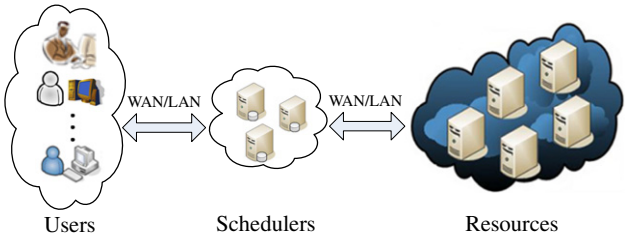


Fig. 1. Resource scheduling in Grid computing.

resources may lead to high communication overhead. Therefore, how to coordinate the scheduling among decentralized schedulers with a moderate communication cost is an important and open problem. A recent work to deal with the above problem has been done in [22], where a collaborative model is proposed based on the Random Early Detection (RED) strategies via gossiping and good scheduling performance is achieved.

Moreover, to meet the need for scheduling adaptation, which comes from the heterogeneity of resources, the variations of resource performance, and the diversity of applications, an adaptive scheduling method is deserved. Recently, a promising approach based on reinforcement learning (RL) has been studied for job scheduling and resource allocation in Grids [23]. As an important class of machine learning methods, RL aims to solve uncertain decision-making problems by interacting with the environment and near-optimal or suboptimal policies can be obtained in a data-driven way [24]. Therefore, RL provides a model-free methodology and is very promising to solve the difficulties of Grid resource scheduling. According to different learning mechanisms, existing RL approaches to resource scheduling can be mainly divided into two types. One is based on policy gradient learning algorithms [6,25,26] and the other uses value-function-based learning algorithms [5,23,27]. However, the learning efficiency and scalability of existing RL methods in Grid resource allocation still need to be improved for large-scale applications of Grid computing.

In this paper, to realize learning-based coordination and generalization in large-scale Grid environments, a novel multi-agent reinforcement learning method, called the ordinal sharing learning (OSL) method, is proposed to solve the job scheduling problem for Grid computing. In the OSL method, a fast distributed learning algorithm is designed based on an ordinal information-sharing mechanism. Compared with previous multi-agent RL (MARL) methods for job scheduling, the OSL method has two aspects of innovations. One aspect simplifies the modeling of optimal decision-making in job scheduling, where only a utility table is learned online to estimate the resources' efficiency, instead of building the complex Grid Information System (GIS). The other aspect circumvents the scalability and coordination problem by an efficient information-sharing mechanism with limited communication for multi-agent systems, where an ordinal sharing strategy makes all agents share their utility tables and make decisions in turn. The proposed approach was evaluated in a simulated large-scale Grid computing environment and the results show its validity and feasibility.

The remainder of this paper is organized as follows. Section 2 introduces a general model for job scheduling in Grid computing, and discusses the performance measures. Section 3 discusses the basic idea of multi-agent reinforcement learning and presents the OSL method for Grid job scheduling. Section 4 makes performance evaluation and comparisons of different job scheduling methods in a simulated Grid computing environment and the results illustrate the effectiveness of the proposed method. Section 5 gives a further overview of the related works. Finally, conclusions are made in Section 6.

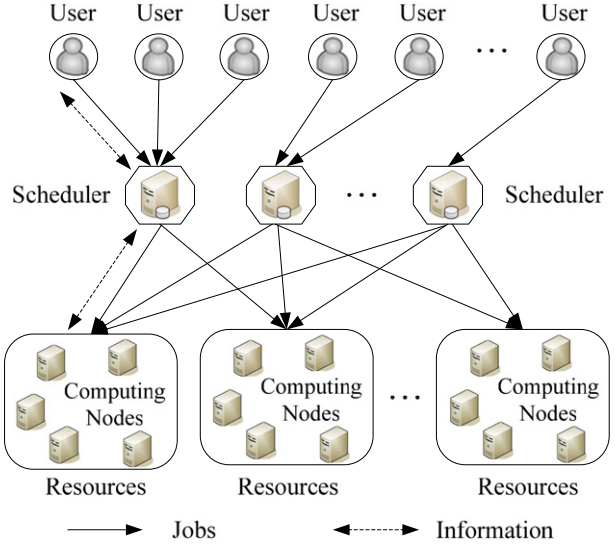


Fig. 2. A general model for Grid job scheduling [22].

2. Problem statement

2.1. A general job scheduling model in Grids

It is well known that the complexity of a general centralized scheduling problem is NP-Complete [28]. Due to the NP-Complete nature and the difficulty to prove the optimality of scheduling algorithms in Grid scenarios, current research always tries to find suboptimal solutions. Moreover, in this paper, to solve the scalability problem, a strategy where decentralized schedulers take charge of job scheduling simultaneously instead of a centralized scheduler is considered. To describe the dynamicity, randomness, heterogeneity of Grid computing, a general Gird job scheduling model is studied, which has been widely used in the literature to evaluate various job scheduling algorithms [22,29,30]. The general job scheduling or resource allocation model for Grids can be illustrated by Fig. 2 [22].

In Fig. 2, the main components include users, schedulers and resources, where different schedulers deal with the jobs in parallel. They are responsible for receiving jobs from users and allocating them to resources. Unlike their counterparts in traditional parallel and distributed systems, Grid schedulers usually cannot control resources directly, but work like brokers [17,31]. Each of the schedulers can submit jobs to any of the computing resources, and finally generate job-to-resource mappings. In the above model, the users merely produce and submit jobs to schedulers and their roles can be replaced by job creators entirely. The model of the job arrival or work loads can be described by a Poisson process or other probability-based models, or by autocorrelation models [30]. In a large-scale Grid system, due to the lack of control over the resources and the long update cycles for resources, the available resource information for schedulers is time delayed and may be inaccurate. Therefore, the limited observability of job schedulers becomes a barrier for job scheduling algorithms based on timely and accurate information, and it is necessary to develop more robust and adaptive scheduling algorithms, which is one of the main motivations of this paper.

In general, the decentralized job scheduling problem in Grids can be modeled as a multi-agent job scheduling system [29], which is denoted as a 6-tuple $\langle G, R, P, D, C, SR\rangle$, where $G = \{g_1, \ldots, g_N\}$ is a set of agents, $S = \{s_1, \ldots, s_M\}$ is a set of resources, $P : G \times N \rightarrow [0, 1]$ is a job submission function, $D : G \times N \rightarrow \Re$ is a probabilistic job size function, $C : G \times N \rightarrow \Re$ is a probabilistic capacity function, and SR is a job scheduling rule.

To focus on the job scheduling tasks, the above model makes some abstraction but maintains the main features of Grid computing environments, i.e., heterogeneity of dynamic, large-scale populations of users and resources [5]. Although the design issues for real implementation, such as network topology, are not considered in detail, the model in Fig. 2 is general enough since different representative models including stochastic job submission functions, job size functions, and capacity functions, can be developed to describe the salient properties of Grid workloads [30]. By making use of stochastic or autocorrelation models of Grid workloads, it is possible to study job scheduling algorithms for dealing with the dynamicity, randomness, heterogeneity of Grid job scheduling problems. In the following, to facilitate discussions, it is assumed that all the schedulers use the same scheduling algorithm and all the jobs only require CPU resources so that they are uniquely characterized by their duration J.

2.2. Performance measures for job scheduling in Grids

In the above Grid scheduling model, resources execute the assigned jobs and may differ in their capabilities, e.g. one computing resource may take more time than the other in executing the same job. Each resource is characterized by its processing capacity C, which is defined as the inverse of CPU time needed to complete a job of a unit length, i.e. if a resource needs duration t to complete a unit job of length $J = 1$, its capacity is $C = 1/t$. Furthermore, it is assumed that all the jobs in the queue are prioritized by their arrival time, hence there is only one single job being executed on a resource at a given time, while the others are waiting in the queue.

The common performance measure in Grid job scheduling is the time-per-token in average (ATPT). The time-per-token (TPT) is measured by the time elapsed between the generation and completion of a job, so the corresponding criterion in average, i.e. ATPT, can be formulated as follows:

$$\mathrm{ATPT} = \frac{1}{L}\sum_{i=1}^{L}\mathrm{TPT}^i = \frac{1}{L}\sum_{i=1}^{L}\left(t^i_{\mathrm{wait}} + t^i_{\mathrm{execute}}\right) \tag{1}$$

where TPT^i is the total elapsed time for the ith job, which is the sum of the queue waiting time t^i_{wait} (the elapsed time between a job submission and the start of execution) and the actual execution time t^i_{execute}, L denotes the total number of jobs completed by all resources. However, ATPT could not characterize the scheduling performance for the whole Grid system in time because only after the job has been completed, could this metric be updated. If the job queues in the resources are very long, the update of ATPT is delayed seriously. Finally, the value of ATPT merely reflects the past efficiency of job scheduling, not the current one.

Therefore, another competent metric, namely the load of resources (LoR, or makespan) is used. The LoR is defined as the total length of the jobs in the queue l_{total} divided by the current resource's capacity C_i, and the average LoR (ALoR) of the system can replace the average time-per-token completely. ALoR can be expressed as:

$$\begin{aligned}\mathrm{ALoR} &= \frac{1}{M}\sum_{i=1}^{M}\mathrm{LoR}^i = \frac{1}{M}\sum_{i=1}^{M}\left(l^i_{\mathrm{total}}/C_i\right)\\ &= \frac{1}{M}\sum_{i=1}^{M}\left(\sum_{j=1}^{L^i}J^i_j/C_i\right)\end{aligned} \tag{2}$$

where LoR^i is the load of the ith resource, l^i_{total} is the total length of the jobs in the queue, which is the sum of all queued jobs' length J^i_j, L^i is the number of jobs in the ith resource's queue, and M is the number of resources. The merit for the new performance measure is evident since it reflects the system performance promptly and comprehensively. Finally, the objective of job scheduling algorithms is to minimize the ALoR and its standard deviation. Minimizing both of the above two quantities will ensure the overall system efficiency as well as fairness. Besides the above two metrics, the maximal LoR in resources is another measure to reflect the transient performance.

3. The OSL method for adaptive job scheduling

As mentioned above, in practical large-scale Grid applications, even with the help of the GIS system, the information about resources in the schedulers is time delayed and potentially inaccurate. So it is reasonable to develop a robust scheduling algorithm which is not dependent on an accurate model. To satisfy the requirements in adaptive job scheduling, a coordinated multi-agent reinforcement leaning method may be an appropriate solution. In the following, after an analysis on different MARL frameworks, a novel decentralized MARL method is proposed for resource selection and job scheduling, where cooperative control among multiple agents or schedulers is achieved by an ordinal sharing learning method.

3.1. Basic frameworks for multi-agent reinforcement learning

Reinforcement learning techniques [24] address the problem of how an agent can learn to approximate an optimal or near-optimal behavioral strategy while interacting with its environment. Most single-agent RL algorithms are based on the formalism of Markov Decision Processes (MDPs) [32]. However, as an extension of RL to distributed decision-making environments, multi-agent reinforcement learning has to deal with the problem that the coexistence of multiple agents breaks the stationary property of the environment. Until now, many MARL algorithms have been developed based on the Stochastic Game (SG) model [33], for example, the JAL [34] and the Team-Q algorithm [35]. Nevertheless, the poor scalability and low information-utilization efficiency are two major obstacles for successful applications of MARL to large-scale applications. For the job scheduling problem depicted in Fig. 2, the number of schedulers and resources is very large, so, it is difficult for previous MARL methods to be adopted.

In addition to the SG model, another framework of MARL is to extend single-agent reinforcement learning techniques to multi-agent systems directly, which is to make each agent learn independently according to the local state and local reward without explicit communication. This technique is called the independent learner (IL) approach in MARL [34] and there have been some IL algorithms developed in the literature [5,36,37]. Although the IL approach to MARL does not need to explore the exponentially growing joint state–action space, the environment will not be stationary any more and there will be convergence problems and oscillatory behaviors in MARL. As we will illustrate in Section 4.1, if the IL method without coordination and communication is used in job scheduling in Grids, a *Herd Behavior* will usually arise [22].

To deal with the above difficulties, a promising approach to multi-agent reinforcement learning is to perform local learning with information sharing and coordination in order to realize the balance between efficiency and optimality. Based on this idea, a novel MARL method called the OSL algorithm will be presented in the following discussions.

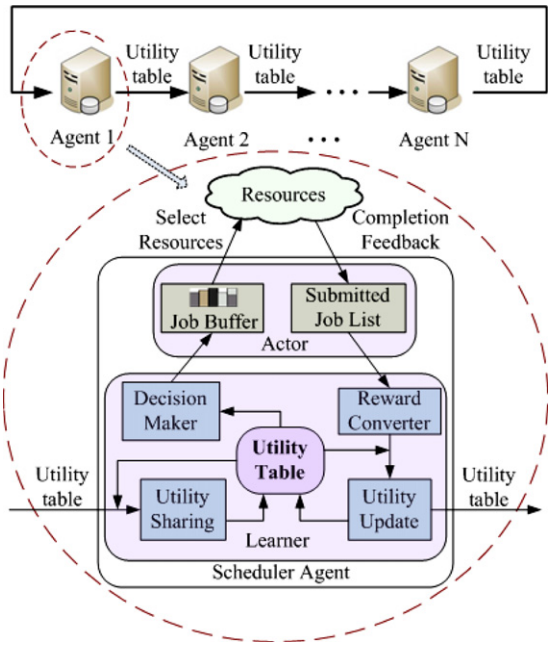


Fig. 3. The schematic diagram of the OSL method for job scheduling.

3.2. The OSL algorithm for job scheduling in Grids

To overcome the 'curse of dimensionality' problem in MARL, we propose the OSL algorithm with decreased computational complexity and an improved coordination mechanism. The new algorithm has two main characteristics. Firstly, it adopts a distributed RL framework and employs a novel utility-table-based learning strategy. Since the OSL method merely utilizes the local information for learning, it is a RL method based on independent learners. Secondly, it makes use of an information-sharing mechanism with limited communication costs to solve the multi-agent coordination problem.

The scheme of OSL is depicted in Fig. 3. The upper loop denotes the schedulers which share the utility tables. The utility table only scales up with the number of resources, M, linearly, so the communication cost is limited. The lower part denotes a scheduler agent in detail. Each scheduler agent mainly comprises two parts: the *Learner* and the *Actor*. The *Learner* receives and shares the utility table from preceding agents in an ordinal manner, and makes decisions to select resources for the jobs queued in the *Job Buffer*. The *Reward Converter* can analyze and convert the jobs' completion signal into reward signals, which are essential to update the utility table. The *Actor* receives the new jobs and arranges them so that they queue up in *Job Buffer*, then submits them to corresponding resources according to the *Learner*'s decisions and records the submission in the *Submitted Job List*. Finally, the *Actor* updates the *Submitted Job List* according to the jobs' completion, namely, if a job is completed, then it will be deleted from the *Submitted Job List*.

Generally speaking, there are two key problems to be considered for the implementation of OSL:

Firstly, although it is well known that the global resource state is the basis for schedulers' decisions, it is hard to obtain accurate information for all schedulers in a dynamic environment due to the schedulers' limited capacities in observation and communication. In this paper, an indirect method, which makes use of the jobs' information to estimate the state, is proposed. A scheduler records information of a submitted job as a vector (n_r, t_s, t_e, J), i.e., the name n_r of the resource used, the job starting time t_s, the job-completion time, t_e, and the job size, J. Then, it abstracts the information to estimate the corresponding resource's state. However, if a scheduler has never submitted jobs to a resource, it knows nothing about it. So, such an estimation from individual experiences merely contains partial information of the global state. To improve the estimation accuracy, some effective means, such as information sharing, must be taken.

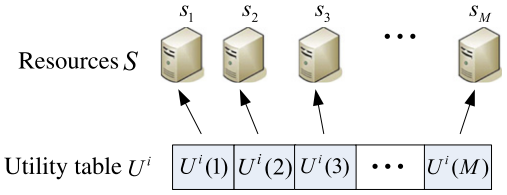


Fig. 4. The utility table U^i.

Secondly, it is hard to obtain the instantaneous rewards for learning directly. The environment cannot provide any global reinforcement signals directly but only individual job-completion signals. So the scheduler agents have to convert such information into reward information. In fact, due to the existence of other schedulers, one job's time-per-token is determined by all schedulers' policies together. How to compute appropriate reinforcement signals from the above information will be a problem. What is more, when one scheduler waits for feedback of its submitted jobs, the Grid environment may change due to the other schedulers' operation, so it is too late for one scheduler to update its utility table only after the job's completion. A possible solution is to develop a reward mechanism to create reward information at every time step, no matter whether the scheduler executed a job submission.

In the following subsections, to solve the above problems, a novel reward generation mechanism and an information-sharing mechanism will be presented.

3.2.1. The decentralized learning strategy using utility tables

In the above model, the scheduler agents are depicted as $G = \{g_1, g_2, \ldots, g_N\}$, where each scheduler agent g_i may take charge of job scheduling for several users. The resources are denoted by $S = \{s_1, s_2, \ldots, s_M\}$. Similar to the learning methods for the bandit problem [24], agent g_i keeps a utility table U^i to score the efficiency of the resources, where $U^i(j)$ denotes the efficiency of the jth resource, or scores the action selecting the jth resource from the resource set S, i.e., $j \in \{1, 2, \ldots, |S|\} = \{1, 2, \ldots, M\}$. The utility table for scheduler g_i is depicted in Fig. 4.

For each time step in the decentralized learning process, the agent g_i executes the resource selection operation and the utility update operation based on the following two steps:

Step 1: The agent g_i checks and judges whether there is a new job arrival. If not, go to *Step* 2. If so, execute *step* 1 repeatedly till all the jobs are scheduled. Agent g_i chooses the resource s_j which has the highest score, then submits the job to resource s_j and records it as an unfinished job in *Submitted Jobs List*. If the jth action is executed, an instantaneous reward is obtained as $r(j) = -1$, and the corresponding utility $U^i(j)$ for this action is updated simultaneously:

$$U^i(j) = (1 - \alpha) * U^i(j) + \alpha * r(j) \tag{3}$$

where α is the learning rate.

Step 2: The agent g_i advances the vacant scheduling process and updates the utilities. As mentioned above, an instantaneous reward signal is designed for each step even if there is no job submission. The agent g_i creates the reinforcement signal for each action according to the jobs' states in the *Submitted Jobs List*, namely:

$$u(j) = \begin{cases} +1 & \text{only the job is finished} \\ 0 & \text{no job} \\ -1 & \text{job is unfinished} \end{cases} \qquad j \in \{1, 2, \ldots, M\}. \tag{4}$$

If multiple jobs are submitted to the same resource, there will be an independent reinforcement signal for each job. Finally, the reward for corresponding actions can be calculated by adding up all the signals:

$$r(j) = \sum_{k=1}^{K^j} u(k) \tag{5}$$

where K^j denotes the number of jobs being submitted to the jth resource currently. For example, suppose that the current agent submitted 3 jobs to the 1*st* resource, and after one time step, a job is finished with the other two jobs unfinished. So the instantaneous reward for selecting the 1*st* resource is: $r(1) = 1+2*(-1) = -1$. When the whole reward vector $(r(1), r(2), \ldots, r(N))$ is obtained, the utility for each resource can be updated with Eq. (3).

By this time, the agent g_i can use the utility table U^i to estimate the efficiency of all the resources. For instance, if one resource's queue is long or the resource's capacity is poor, after the scheduler submitted a job to it, the scheduler has to wait for a long time to receive the accomplished response from the resource. So, the scheduler will get the reward signal -1 for much more times than getting reward $+1$. Finally, the corresponding utility value of this resource will be small. Obviously, according to the utility table, the bigger the utility value becomes, the better the resource state is. The successive update operation gives a timely feedback of resources' working state, and it is essential to make a feasible decision for allocating successive jobs.

3.2.2. Multi-agent information sharing based on limited communication

In the above subsection, a utility table is built to estimate the efficiency of the resources. A modified reward and update mechanism is developed to indicate the resources' efficiency. However, in Grid applications, there are multiple scheduler agents. If all the agents learn and make decisions independently and simultaneously, a coordination problem occurs. Apparently, any decision of a scheduler will change the resources' state, but the other schedulers will not detect the change until they submit jobs to the same resource (they indirectly detect this by using the waiting time in the queue). So the utility table in a particular agent cannot exactly indicate the true state of the resources. Furthermore, possible collisions will become more severe along with the increasing number of schedulers. Therefore, a feasible coordination mechanism must be developed for distributed learning in the Grid job scheduling problem.

Since each agent owns a local utility table to estimate the resources' efficiency, it is a natural way to improve the precision of estimation by sharing the utility tables. However, since the number of scheduler agents in Grids is very large, it is impossible to share each utility table directly. Therefore, in this paper, an ordinal sharing mechanism with limited communication is proposed to meet the above requirement. The coordination among agents is realized by sharing neighboring agent's utility table ordinally and iteratively. As shown in Fig. 4, the utility table is merely in proportion to the scale of resources linearly. So the total communication cost among agents is low and is constant at all times.

To implement the information-sharing mechanism, an ordinal structure is defined for all the scheduler agents by sorting the agents as $g_1, g_2, \ldots, g_N$, then the agents share their utility tables and make their decisions in an ordinal manner, namely, agent g_i shares the preceding agents' utility information as follows:

$$U^i(j) = (1-\beta)\cdot U^i(j) + \beta \cdot U^{i-1}(j) \tag{6}$$

where β is the sharing factor. $U^{i-1}(j)$ is the utility table of neighboring agent g_{i-1} and contains all the preceding agents' estimation of resources' efficiency. Finally, the last agent's utility table is returned to the first agent for sharing again. In other words, the utility sharing process is ordinal and iterative.

Table 1 shows the main procedure in the OSL algorithm for agent g_i in Grid job scheduling.

Compared with other MARL algorithms, the OSL algorithm is more suitable to be implemented in large-scale job scheduling applications. As a utility-table-based learning method, there are no explicit state variables in the utility table function, so it is more adaptive to the Grid scenarios where both resources and applications are highly diverse and dynamic. Furthermore, another important advantage of OSL is the low communication costs for coordination. The total amount of information exchange is the simple utility table, whose scale is linear with respect to the number of the resources and is much lower than the exponential one in direct communication mode.

4. Performance evaluation and discussions

In this section, the performance of the *OSL*-based *Selection (OSLS)* rule for job scheduling will be evaluated and analyzed in simulations. In addition, the proposed *OSLS* method is compared with four other resource scheduling or selection rules, which are *Decentralized Min–Min Selection (DMMS)* [38], *Random Selection* (RS), *Least Load Selection (LLS)*, and *Simple Learning Selection (SLS)* [5]. The *Min–Min* algorithm is a heuristic scheduling method that becomes a benchmark scheduling algorithm for performance comparisons [38]. Based on the decentralized scheduling model, each scheduler executes the decentralized *Min–Min* algorithm independently. A scheduler's decision may not be accurately known by others in a dynamic environment even with the help of the GIS system. The reason is that time delays are always inevitable for the information updates in GIS. In the *RS* method, an agent chooses resources for a job randomly according to a uniform probability distribution. In the *LLS* method, an agent chooses the least loaded resources to submit a job. If there are multiple resources with the same minimum load, then one of them is chosen randomly. This selection rule assumes that the agents can obtain accurate global resource information, for instance, from an ideal GIS system. In the *SLS* method, agents perform independent reinforcement learning processes. It is different from the proposed *OSLS* method in that an agent does not update its utility table until it received the final completion signals for the submitted jobs from the resources, and each agent learns independently without any coordination information [5].

The scale of the Grid system can be defined as the combination (N, M) of the number of agents (N) and the number of resources (M). During every time step, each scheduler agent g_i may receive jobs with random numbers generated by a Poisson process. The arriving rate of the jobs is denoted as ξ. The lengths of the jobs are generated randomly from a uniform distribution in interval $[J_{\min}, J_{\max}]$. The capacities of the resources were also chosen uniformly in the interval $[C_{\min}, C_{\max}]$. So the system can be described by the parameter set $(N, M, \xi, [J_{\min}, J_{\max}], [C_{\min}, C_{\max}])$. As a result, the expected system load γ_{system}, which is jointly determined by the total disposal capacity of Grids and the total jobs arrived, can be calculated as:

$$\begin{aligned}\gamma_{\text{system}} &= \frac{J_{\text{total}}}{C_{\text{total}}} = \frac{\sum_{j=1}^{N}(J_j^{av} * \xi_j)}{\sum_{i=1}^{M} C_i} \\ &= \frac{\sum_{j=1}^{N}((J_{\min}+J_{\max})/2 * \xi_j)}{\sum_{i=1}^{M} C_i} * 100\%\end{aligned} \tag{7}$$

Table 1
The OSL algorithm for Grid job scheduling.

Input and initialization:

// $U^i(j) = -1, j = 1,..., M$:initialize all the cells of utility table to –1;

// α : learning rate; β : sharing factor;

// $k = 0$: initialize time step parameter to 0;

// $J_{submitted} = \varnothing$: a set of submitted jobs; $J_{new} = \varnothing$: a set of new jobs .

The Learning Scheduling Loop:

For *each time step* $k \le MaxSteps$, Do:

g_i *receive the utility table* U^{i-1} *from* g_{i-1};

g_i *updates the utility table as follows* :

$U^i(j) \leftarrow (1-\beta) * U^i(j) + \beta * U^{i-1}(j),\ j = 1,..., M$

While $J_{new} \neq \varnothing$, Do :

$job \leftarrow SelectJob(J_{new})$;

$J_{submitted} \leftarrow J_{submitted} \bigcup job$;

$j_{index} \leftarrow Index(\ MaxUtility(\ U^i(j)\ )\ )$;

$SubmitJobToResource(j_{index})$;

$U^i(j_{index}) \leftarrow (1-\alpha) * U^i(j_{index}) + \alpha * (-1)$

$J_{new} \leftarrow J_{new} \setminus job$

End While

g_i *send utility table* U^i *to next agent* g_{i+1};

Execute the simulation operation and update all the jobs' states;

Set $r(j) = 0, j = 1, 2,..., M$

For *each job* $job_{submitted}$ *in* $J_{submitted}$, Do:

$j_{index} \leftarrow IndexJob(job_{submitted})$;

$state \leftarrow CheckJobState(job_{submitted})$;

$$sig(j_{index}) = \begin{cases} 1 & If\ \ state\ =\ Finished \\ -1 & Else \end{cases};$$

$r(j_{index}) \leftarrow r(j_{index}) + sig(j_{index})$

If $state\ =\ Finished$

$J_{submitted} \leftarrow J_{submitted} \setminus job_{submitted}$

End If

End For

For $j = 1, 2,..., M,$ Do:

$U^i(j) \leftarrow (1-\alpha) * U^i(j) + \alpha * r(j)$;

End For

$k \leftarrow k + 1$

End For

where J_j^{av} is the medium value of the jobs' length for the jth scheduler. Apparently, the system load should not be more than 100%, otherwise the scheduling system will go corrupt. In fact, a system load more than 90% is very heavy for Grids. An effective job scheduling algorithm can balance the load of the system by utilizing all the resources fairly and fully. To test the new method's load-balancing capability, several experiments are carried out in what follows.

4.1. Performance evaluations under different system scales

To experiment with different system scales, three kinds of system scales, namely (30, 100), (100, 250) and (300, 1000), are selected. These configurations are large enough to represent the scale of a typical Grid computing environment. The dispersion range of the jobs' length is set as [5, 995]. The interval of resources' capacity is [50, 350]. The job arrival rate is 0.93, 0.7 and 0.93, respectively. So all the system loads are approximately around 70%. It is a typical medium system load for Grid applications. The simulation results are shown in Fig. 5.

In Fig. 5, the ALoR curves of different job scheduling methods show that, under medium system loads, only the OSL method, the DMMS method and the LLS method achieve efficient load balancing in different system scales. Obviously, the LLS method is a centralized method and can achieve the best scheduling policy. But the expensive costs for computing and communication prevent it from being effectively applied in real-world Grids. The DMMS method can balance the load, however, the efficiency is lower than the LLS method in that non-coordinated decisions may conflict and result in over-utilization/under-utilization of some resources. The OSL algorithm is decentralized and needs only a limited communication cost, but can obtain a better suboptimal policy for different system scales. It is shown that, at the beginning, the OSL method may have worse performance than DMMS and LLS. This is because the schedulers using the OSL method have no prior knowledge of the Grids, but the schedulers with the other two heuristic methods can obtain the environment information from the GIS system. When the OSL-based schedulers accumulate enough experiences by trials, a good scheduling performance can be achieved finally.

As is shown, the performance of the SLS rule is very poor and it fails to accomplish the job scheduling task. This phenomenon arises mainly due to the inappropriate reward mechanism and the synchronization problem. For the SLS method, it adopts a delayed reward mechanism regardless of its bandit-like model. Moreover, the SLS-based schedulers learn and work independently without coordination. Evidently, this behavior causes over-utilization of some resources while leading under-utilization of others, which will deteriorate the scheduling performance. This pathology is known as herd behavior [18,19].

The agents with the RS rule choose the resources randomly without considering their efficiency at all, so the LoR on the resources with low capacities will grow indefinitely. Finally, the average load, ALoR, increases out of control. Moreover, for the RS rule, it is true that, the larger the scale is, the worse the performance becomes.

4.2. Performance evaluations under different system loads

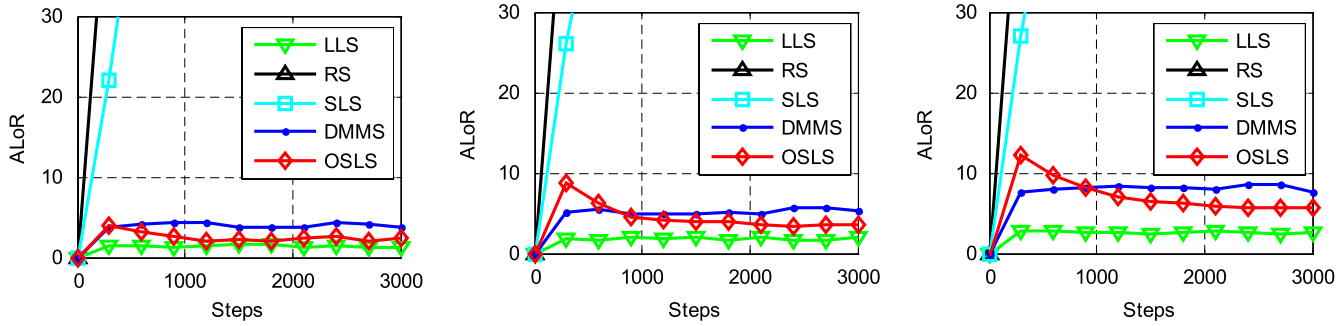
To test the new method's adaptive performance under different system loads, three system load configurations, i.e., 50%, 70% and 90%, are selected, and the system scale is set as (100, 250). The other parameters are the same as the former experiment. Fig. 6 shows the variation curves of ALoR under different system loads.

Clearly, it is shown that the OSL method allows the agents to schedule jobs among the resources much more efficiently than the RS and SLS methods. Moreover, the OSL approach can converge to a suboptimal policy even with increasing system loads. Therefore, the proposed OSL method can be adaptive to different system loads. From Fig. 6, it can be observed that for low system loads, OSL can even converge to a near-optimal policy which has similar performance as the centralized LLS method. When the system loads increase, OSL can also find a suboptimal policy which is comparable to LLS.

4.3. Performance evaluations under different resource capacities

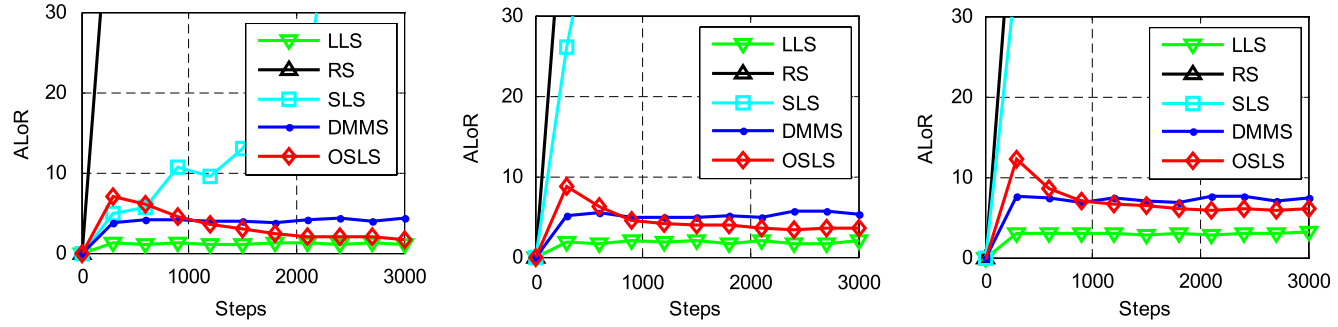
In the former simulations, the interval of resources' capacity is broad, namely [50, 350]. In fact, different intervals can affect the system performance remarkably. In the following, a narrow interval of resources' capacity, [150, 250], is chosen, and the above simulation with system scale (200, 500) is conducted again. The results are shown in Fig. 7.

In Fig. 7, when the system load is light ($\gamma_{\text{system}} \le 70\%$), the RS method is good enough for job scheduling. The reason may be that all resources' capacities, which are all larger than 150, are high



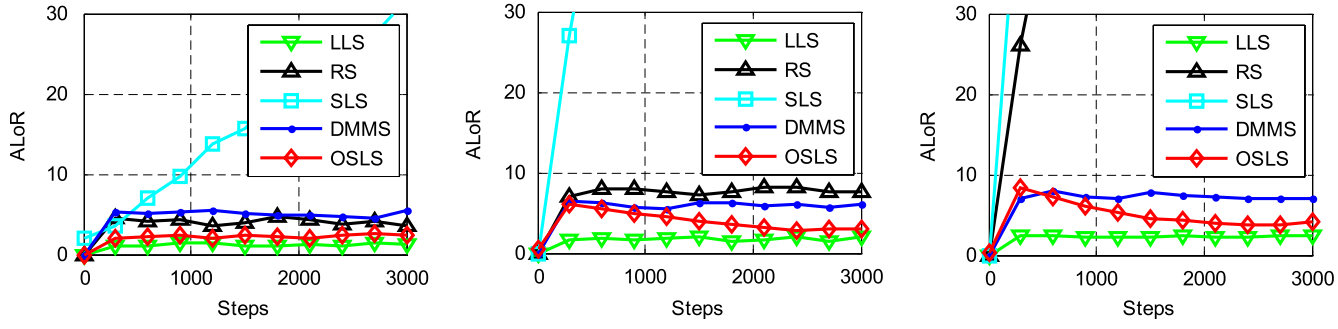
(a) Small scale. (b) Medium scale. (c) Large scale.

Fig. 5. Performance comparisons with different scales under medium system loads.



(a) $\gamma_{system} = 50\%$. (b) $\gamma_{system} = 70\%$. (c) $\gamma_{system} = 90\%$.

Fig. 6. Performance comparisons under different system loads.



(a) $\gamma_{system} = 50\%$. (b) $\gamma_{system} = 70\%$. (c) $\gamma_{system} = 90\%$.

Fig. 7. Performance comparisons with different resource capacities.

enough to avoid keeping a long job queue. In addition, the SLS method can also obtain good performance when the system load is low, which is shown in Fig. 7(a). However, both RS and SLS cannot get good performance for a high system load, as shown in Fig. 7(b) and (c). When the system load increases, the performance of the RS method declines and becomes infeasible finally. However, the OSL method can achieve load balancing consistently with different resource capacities.

4.4. Performance evaluations under different numbers of schedulers and resources

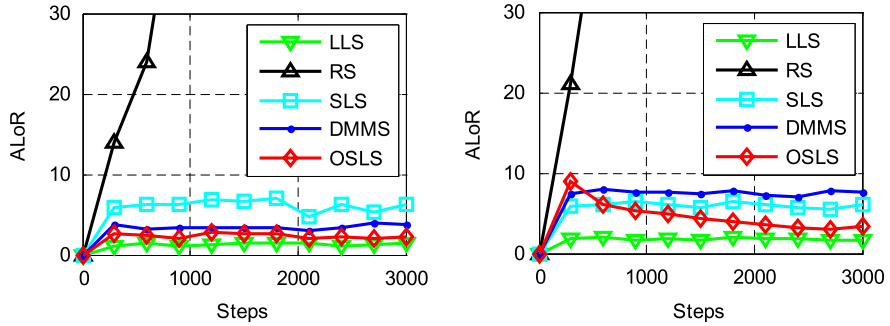
In the following simulations, different ratios of scheduler numbers and resource numbers are chosen as (500, 200) and (1000, 500). The interval of jobs' length and the interval of resources' capacity are [5, 995] and [250, 750], respectively. The job arrival rate is 0.2 and 0.4 (so the system load is 50% and 80%, respectively). The results are shown in Fig. 8.

According to Fig. 8, it is obvious that the SLS method has good performance when the number of schedulers is much greater than the number of resources. The results are consistent with the conclusion in [5]. Such good results may be due to the low job arrival rate and the powerful resource capacities. However, for real-world Grids, it is not common for the number of schedulers to be more than the number of resources. Moreover, notice that the performance of the SLS rule is still inferior to the OSL method under different conditions.

4.5. Other performance measures

Besides the average load of resource (ALoR), other metrics can be used to measure the system performance. Generally, ALoR indicates the macro-performance of the system, but the transient performance, such as the maximal LoR (or makespan) in the resources, is also important. Moreover, the standard deviation of the LoR is another metric to evaluate the efficiency of a job scheduling algorithm. Figs. 9 and 10 show the corresponding results of the experiments in Section 4.1.

From the above two figures, it is found that the maximal LoR and the deviation for OSLS are convergent while the RS, SLS and



(a) $\gamma_{\text{system}} = 50\%$. (b) $\gamma_{\text{system}} = 80\%$.

Fig. 8. Performance comparisons with different number of schedulers and resources.

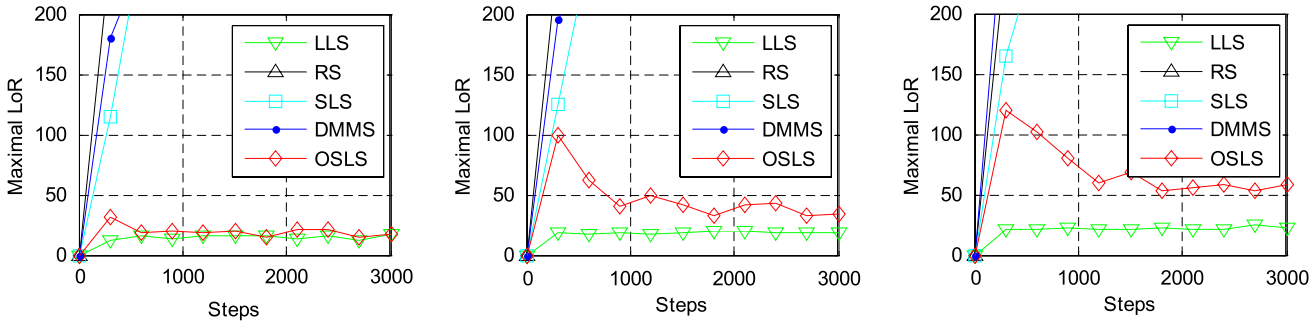


Fig. 9. Maximal LoR under small, medium and large system scales.

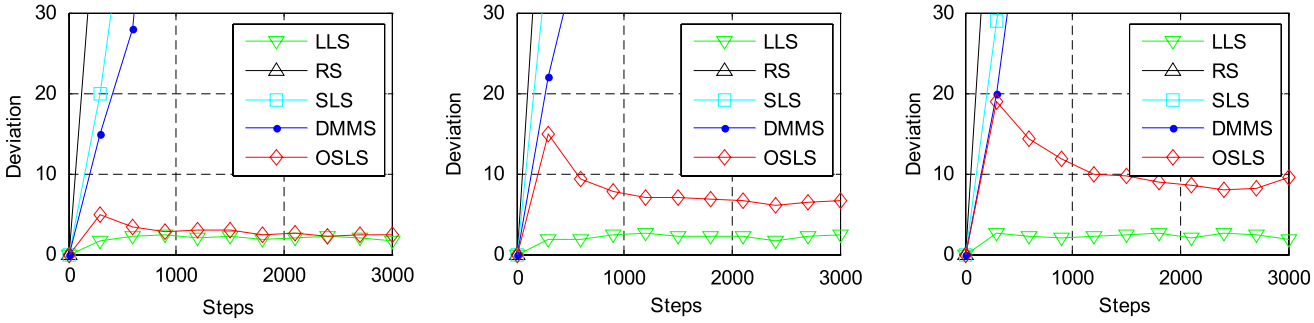


Fig. 10. Standard deviation of LoR under small, medium and large system scales.

DMMS rules are divergent quickly. Although the magnitudes of the OSLS curves increase along with the increasing system scale, they level off finally. The same results were obtained under different system loads. In other words, all the results show that the transient performance and efficiency of the OSL algorithm is satisfactory.

4.6. Performance evaluation with different learning rates and sharing factors

In all the above simulations, the learning rate and the sharing factor are both set as 0.5. In fact, the learning factors and sharing factors can be viewed as compromises between new information and past experiences, as well as an agent's own knowledge and others' knowledge. Some different configurations for the two parameters can be chosen. In the following, an experiment, which has the same conditions as the one in Section 4.1 (b), is made, where different learning rates and sharing factors are evaluated, respectively.

Fig. 11(a) shows the performance variations of OSL with different learning rates, 0.2, 0.5 and 0.8, respectively, where the sharing factor is equal to 0.5. Fig. 11(b) shows the performance variations of OSL with different sharing rates, 0.2, 0.5 and 0.8, respectively, where the learning rate is equal to 0.5. The results show that too large or small values of α and β may result in undesired scheduling performance. From the empirical studies, a medium value of the learning rates and the sharing factors can be selected to get good performance.

4.7. Summary

According to the above experiments, the advantages of the OSL algorithm are obvious. By using the ordinal sharing learning mechanism, the OSL algorithm achieves a comparable performance to a centralized and model-based method, i.e. the LLS method, but with much lower computational costs and limited communication. Furthermore, the new method is less sensitive to working conditions than other algorithms and achieves effective load balancing based on MARL. Table 2 shows a summary of the comparisons among different job scheduling methods studied in this paper.

5. Related works

For the RL-based job scheduling problem in Grids, there are some other related works. In [5], the SLS method was adopted

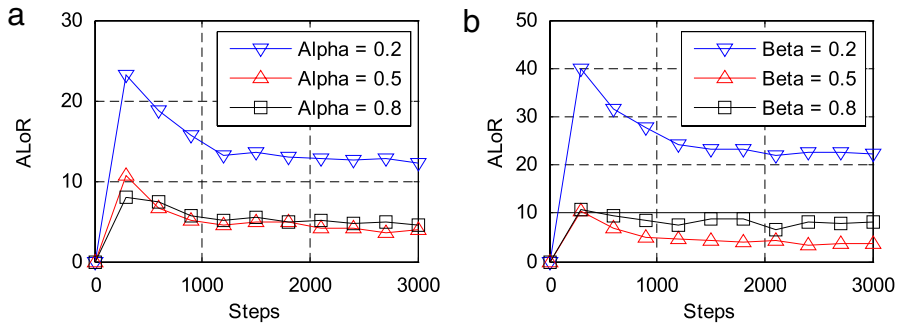


Fig. 11. Performance of OSL with different learning rates (left) and sharing factors (right).

Table 2
Comparisons among different job scheduling methods.

	Structure	Learning	Communication	Convergence	Optimality
RS	Distributed	No	No	No	No
LLS	Centralized	No	Heavy	Yes	Near-optimal
SLS	Distributed	Yes	No	No	No
DMMS	Distributed	No	Delayed/heavy	Yes	Suboptimal
OSLS	Distributed	Yes	Light	Yes	Suboptimal

for Grid job scheduling. However, the above experimental results show that the SLS method only has good performance in some special cases when the number of users is much more than the number of resources. Moreover, its performance still needs to be improved.

In [6], the authors introduced a new gradient ascent learning algorithm named Weighted Policy Learner (WPL) for the distributed task allocation problem in domains like the Grids, and other distributed systems. WPL can learn a stochastic policy without observing other agents' actions. However, for the limited observation information and the difficulty in acquiring an equilibrium solution, the convergence rate of multi-agent gradient ascent methods is slow, especially for large-scale problems. So, the authors merely tested WPL for a small-scale problem, where both the numbers of severs and users are no more than five.

To solve the coordinated learning problem in dynamic resource allocation, some value-function-based RL algorithms were proposed in [39,27]. To extend standard Q-learning to resource allocation problems with large or continuous state–action spaces, RL methods with function approximation have been studied. However, it is still hard to solve the problem of large-scale Grid applications. In [25,26], a multi-agent RL approach named the Fair Action Learner (FAL) algorithm was applied to share the resources across clusters in a decentralized manner. FAL adopts a direct policy search technique, the Policy Gradient Ascent (PGA) algorithm, to learn decision-making policies. However, from their experimental results, the convergence rate of the learning process is still slow.

In [40], the authors treated the resource allocation problem as a composite MDP and proposed a simplified localized RL approach where the actions, states and rewards are all absolutely localized. The local RL approach was tested in the resource allocation task for a data center prototype and some promising results were obtained. However, proper coordination among agents is essential to get better system performance.

6. Conclusions

One of the key concerns of Grid computing is to develop autonomic computing systems that have the abilities of self-configuration and self-optimization in dynamic environments. In this paper, the OSL method based on multi-agent reinforcement learning is proposed to solve the job scheduling problem in Grids. This approach circumvents the scalability problem by using a distributed learning strategy, and achieves multi-agent coordination based on an ordinal information-sharing mechanism. Finally, the performance of the OSL algorithm is evaluated and compared with other algorithms, where a general Grid job scheduling model is studied and simulated to describe the dynamicity, randomness, heterogeneity of Grids. The simulation results illustrate that a proper online learning method can have a substantial positive effect on the quality of load balancing in a heterogeneous Grid system, and the effectiveness and efficiency of the OSL algorithm is illustrated. Future work may include the improvement and application of the proposed method in real Grid environments.

Acknowledgements

This work is supported in part by National Natural Science Foundation of China under Grants 60774076 and 61075072, the Fork Ying Tong Youth Teacher Foundation Under Grant 114005, and Natural Science Foundation of Hunan Province under Grant 07JJ3122. We also thank the anonymous reviewers for their comments and recommendations, which have been crucial to improving the quality of this work.

References

[1] Y. Chevaleyre, P.E. Dunne, U. Endriss, J. Lang, M. Lemaître, N. Maudet, J. Padget, S. Phelps, J.A. Rodríguez-Aguilar, P. Sousa, Issues in multi-agent resource allocation, Informatica 30 (1) (2006) 3–31.
[2] R. Feldmann, M. Gairing, T. Lücking, B. Monien, M. Rode, Selfish Routing in Non-Cooperative Networks: A Survey, Springer-Verlag, 2003, pp. 21–45.
[3] E. Cantillon, M. Pesendorfer, Auctioning bus routes: the London experience, in: P. Cramton, et al. (Eds.), Combinatorial Auctions, MIT Press, 2006.
[4] P. Gradwell, J. Padget, Markets vs. auctions: approaches to distributed combinatorial resource scheduling, Journal Multiagent and Grid Systems 1 (4) (2005) 251–262.
[5] A. Galstyan, K. Czajkowski, K. Lerman, Resource allocation in the grid with learning agents, Journal of Grid Computing 3 (2005) 91–100.
[6] S. Abdallahy, V. Lesser, Learning the task allocation game, in: Proceedings of the Fifth AAMAS, Japan, May 8–12, 2006, pp. 850–857.
[7] I. Foster, C. Kesselman (Eds.), The Grid: Blueprint for a New Computing Infrastructure, 2nd ed., Morgan Kaufmann, 2004.
[8] F.P. Dong, S.G. Akl, Scheduling algorithms for grid computing: state of the art and open problems, Technical Report No. 2006-504, School of Computing, Queen's University, Kingston, Ontario, Canada, January 2006.
[9] F. Xhafa, A. Abraham, Computational models and heuristic methods for grid scheduling problems, Future Generation Computer Systems 26 (2010) 608–621.

[10] W.C. Chung, R.S. Chang, A new mechanism for resource monitoring in grid computing, Future Generation Computer Systems 25 (2009) 1–7.
[11] R.S. Chang, J.S. Chang, P.S. Lin, An ant algorithm for balanced job scheduling in grids, Future Generation Computer Systems 25 (2009) 20–27.
[12] D. Thain, T. Tannenbaum, M. Livny, Distributed computing in practice: the Condor experience, Concurrency and Computation: Practice and Experience 17 (15) (2005) 323–356.
[13] Portable Barch System, 2009. http://www.pbsgridworks.com/.
[14] Sun Grid Engine, 2009. http://www.sun.com/software/gridware/.
[15] K. Krauter, R. Buyya, M. Maheswaran, A taxonomy and survey of grid resource management systems for distributed computing, Software: Practice and Experience 32 (2) (2002) 135–164.
[16] J. Frey, T. Tannenbaum, M. Livny, I. Foster, S. Tuecke, Condor-*G*: a computation management agent for multi-institutional grids, in: Proc. of the 10th IEEE Int'l. Symp. on High Performance Distributed Computing, HPDC-10, 2001, pp. 237–246.
[17] F. Berman, R. Wolski, H. Casanova, et al., Adaptive computing on the grid using AppLeS, IEEE Transactions on Parallel and Distributed Systems (TPDS) 14 (4) (2003) 369–382.
[18] W. Cirne, F. Berman, When the herd is smart: aggregate behavior in the selection of job request, IEEE Transactions on Parallel and Distributed Systems (TPDS) 14 (2) (2003) 181–192.
[19] Q. Zheng, H. Yang, Y. Sun, How to avoid herd: a novel stochastic algorithm in grid scheduling, in: Proc. of the 15th IEEE Symp. on High Performance Distributed Computing, 2006, pp. 267–278.
[20] J.B. Weissman, A. Grimshaw, Federated model for scheduling in wide area systems, in: Proc. of the 5th IEEE Int'l Symp. on High Performance Distributed Computing, 1996, pp. 542–550.
[21] A.R. Butt, R. Zhang, Y.C. Hu, A self-organizing flock of condors, Journal of Parallel and Distributed Computing 66 (1) (2006) 145–161.
[22] M. Brugnoli, E. Heymann, M.A. Senar, Grid scheduling based on collaborative random early detection strategies, in: 18th Euromicro Conference on Parallel, Distributed and Network-Based Processing, 2010, pp. 35–42.
[23] G. Tesauro, N.K. Jong, R. Das, Mohamed, N. Bennani, A hybrid reinforcement learning approach to autonomic resource allocation, in: ICAC 06, Dublin, Ireland, June, 2006, pp. 65–73.
[24] R.S. Sutton, A.G. Barto, Reinforcement Learning: An Introduction, MIT, Cambridge, MA, 1998.
[25] C.J. Zhang, V. Lesser, P. Shenoy, A multi-agent learning approach to resource sharing across computing clusters, Technical Report, 2008.
[26] C.J. Zhang, V. Lesser, P. Shenoy, A multi-agent learning approach to online distributed resource allocation, in: Proceeding of the 21st International Joint Conferences on Artificial Intelligence, IJCIA2009, 2009, pp. 361–366.
[27] D. Vengerov, A reinforcement learning approach to dynamic resource allocation, Engineering Applications of Artificial Intelligence 20 (3) (2007) 383–390.
[28] H. El-Rewini, T. Lewis, H. Ali, Task Scheduling in Parallel and Distributed Systems, PTR Prentice Hall, 1994.
[29] A. Schaerf, Y. Shoham, M. Tennenholtz, Adaptive load balancing: a study in multi-agent learning, Journal of Artificial Intelligence Research 2 (1995) 475–500.
[30] H. Li, R. Buyya, Model-based simulation and performance evaluation of grid scheduling strategies, Future Generation Computer Systems 25 (2009) 460–465.
[31] I. Rodero, F. Guim, J. Corbalan, L. Fong, S. Masoud Sadjadi, Grid broker selection strategies using aggregated resource information, Future Generation Computer Systems 26 (2010) 72–86.
[32] M.L. Puterman, Markov Decision Problems, Wiley, NY, 1994.
[33] L. Shapley, Stochastic games, Proceedings of the National Academy of Sciences 39 (1953) 1095–1100.
[34] C. Claus, C. Boutilier, The dynamics of reinforcement learning in cooperative multiagent systems, in: Proceedings of the Fifteenth National Conference on Artificial Intelligence, 1998, pp. 746–752.
[35] M.L. Littman, Value-function reinforcement learning in Markov games, The Journal of Cognitive Systems Research 2 (1) (2001) 55–66.
[36] Y. Wang, C.W. De Silva, Multi-robot box-pushing: single-agent *Q*-learning vs. team *Q*-learning, in: Proceedings of the 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS, Beijing, China, 2006, pp. 3694–3699.
[37] M.J. Matarić, Reinforcement learning in the multi-robot domain, Autonomous Robots 4 (1) (1997) 73–83.
[38] R.F. Freund, M. Gherrity, S. Ambrosius, et al. Scheduling resources in multi-user, heterogeneous, computing environments with SmartNet, in: Proc. the 7th IEEE Heterogeneous Computing Workshop, HCW'98, Orlando, FL, USA, March 1998, pp. 184–199.
[39] D. Vengerov, Multi-agent learning and coordination algorithms for distributed dynamic resources allocation, Ph.D. Dissertation, Stanford University, 2004.
[40] G. Tesauro, Online resource allocation using decompositional reinforcement learning, in: Proceedings of AAAI-05, 2005, pp. 886–891.

Jun Wu received the B.Sc. and M.Sc. degrees in electrical engineering from National University of Defense Technology, Changsha, China, in 2002 and 2005, respectively. He is currently working toward the Ph.D. degree from the Institute of Automation, National University of Defense Technology, China. His current research interests include reinforcement learning, autonomous agent and multi-agent systems, especially in resource allocation, and multi-robot control.

Xin Xu received the B.S. degree in control engineering from the Department of Automatic Control, National University of Defense Technology (NUDT), Changsha, PR China, in 1996 and the Ph.D. degree in electrical engineering from the College of Mechatronics and Automation (CMEA), NUDT. From 2003 to 2004, he was a Postdoctoral Fellow at School of Computer, NUDT. In August, 2006 and from September to October 2007, he was a visiting scholar for cooperation research in the Hong Kong Polytechnic University, Hong Kong, China and the University of Strathclyde, UK, respectively. Currently, he is an Associate Professor at the Institute of Automation, CMEA, NUDT.

He has coauthored four books and published more than 50 papers in international journals and conferences, including IEEE Transactions on Neural Networks, Journal of AI Research, etc. His research interests include reinforcement learning, data mining, learning control, robotics, autonomic computing, and computer security.

Dr. Xu received the excellent Ph.D. dissertation award from Hunan Province, PR China, in 2004 and the Fork Ying Tong Youth Teacher Fund of China in 2008. He has served as a PC member or Session Chair in many international conferences, and currently, he is a reviewer for several journals including several IEEE Transactions. He has been a grant reviewer of National Natural Science Foundation of China since 2005.