Applied Soft Computing 52 (2017) 376–386

Contents lists available at ScienceDirect

Applied Soft Computing

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

Learning non-cooperative game for load balancing under self-interested distributed environment

Zheng Xiao[a,*], Zhao Tong[b], Kenli Li[a], Keqin Li[a,c]

[a] College of Information Science and Engineering, Hunan University, Changsha, People's Republic of China
[b] College of Mathematics and Computer Science, Hunan Normal University, Changsha, People's Republic of China
[c] Department of Computer Science, State University of New York, NY 12561, USA

ARTICLE INFO

Article history:
Received 4 May 2014
Received in revised form 22 July 2016
Accepted 21 October 2016
Available online 26 October 2016

Keywords:
Distributed computing
Job scheduling
Load balancing
Non-cooperative game
Reinforcement learning

ABSTRACT

Resources in large-scale distributed systems are distributed among several autonomous domains. These domains collaborate to produce significantly higher processing capacity through load balancing. However, resources in the same domain tend to be cooperative, whereas those in different domains are self-interested. Fairness is the key to collaboration under a self-interested environment. Accordingly, a fairness-aware load balancing algorithm is proposed. The load balancing problem is defined as a game. The Nash equilibrium solution for this problem minimizes the expected response time, while maintaining fairness. Furthermore, reinforcement learning is used to search for the Nash equilibrium. Compared with static approaches, this algorithm does not require a prior knowledge of job arrival and execution, and can adapt dynamically to these processes. The synthesized tests indicate that our algorithm is close to the optimal scheme in terms of overall expected response time under different system utilization, heterogeneity, and system size; it also ensures fairness similar to the proportional scheme. Trace simulation is conducted using the job workload log of the Scalable POWERpallel2 system in the San Diego Supercomputer Center. Our algorithm increases the expected response time by a maximum of 14%. But it improves fairness by 12–27% in contrast to Opportunistic Load Balancing, Minimum Execution Time, Minimum Completion Time, Switching Algorithm, and k-Percent Best.

1. Introduction

Grid and cloud computing [1] are two widely deployed large-scale distributed systems. Computing resources that are connected through the Internet can spread worldwide. As far as the number and types of jobs are concerned, distributed systems can provide unimaginable computation capacity by gathering resources as many as possible, and thus undertake a large amount of concurrent requests. Load balancing is the key to exploiting the huge potential of distributed systems.

Different kinds of interactions between resources are involved because multiple autonomous domains exist in large-scale distributed systems. Resources in the same domain generally tend to be cooperative. They share the same goals. Some previous algorithms listed in Section 2 such as Opportunistic Load Balancing (OLB), Minimum Execution Time (MET), Minimum Completion Time (MCT), Switching Algorithm (SA), and k-Percent Best (kPB) can be applied for this cooperative interaction [2–6]. To the contrary, interactions between different domains are usually self-interested. Resources have their own interests or goals. For example, they need to minimize the response time of services they provide, so that sometimes they have to turn down ones that are not their own users. Rao and Kwork [7,8] present some selfish scenarios in grid and model them using game theory.

Considering the emergence of cooperative and self-interested interactions, resources in distributed systems can be classified into three groups as per their roles. The first class, which is dedicated for computation, is called processing elements (PEs) in this study. The second class is homo-schedulers, which is the bridge to achieve full cooperation among PEs. Through cooperative interaction, homo-schedulers unite all PEs that are affiliated to them, to finish the common goal. The last class, called heter-schedulers, is independent and self-interested. Their own benefits have a priority. They are often located in different domains. The interaction among different classes should follow different protocols, i.e., different load balancing schemes.

In this study, we focused on the interaction among heter-schedulers. We propose a load balancing scheme for

* Corresponding author.
E-mail addresses: zxiao@hnu.edu.cn (Z. Xiao), tongzhao1985@yahoo.com.cn (Z. Tong), lkl510@263.net (K. Li), lik@newpaltz.edu (K. Li).

http://dx.doi.org/10.1016/j.asoc.2016.10.028
1568-4946/

large-scale distributed systems, achieving collaboration under a self-interested environment.

Heter-schedulers have their own objectives. They are only willing to accept some of the jobs that will not negatively influence their performance. If the load assignment is unfair, some resources have to contribute significantly more than others. Self-interested resources undoubtedly have no incentive to make such sacrifice. Therefore, fairness is the key to collaboration among heter-schedulers.

The policy of one heter-scheduler depends on the policies of others. Thus, this problem can be modeled as a non-cooperative game. The Nash equilibrium solution for this problem minimizes the expected response time, while maintaining fairness. Similar to a prisoners' dilemma in game theory, each participant attempts to minimize their own response time until no one can profit from strategy alteration. Finally all the participants have an equal response time, which leads to fairness.

To find the Nash equilibrium solution, which ensures fairness, some static scheduling algorithms are proposed [9–11]. They use queueing theory to estimate the utilities of each allocation. Refs. [9,11] are based on M/M/1 model while M/G/1 in [10]. In these models, job arrival and service rates are assumed to be a priori knowledge. Though, job arrival and execution processes are unpredictable in a distributed system. Estimating the parameters is a non-trivial task, and the exactness of the model remains suspicious. In contrast, reinforcement learning, an online unsupervised learning method, is used in our algorithm. It does not require a priori knowledge of job arrival and execution, and adapt dynamically to these processes based on online samples, which is effective and practical. When all the heter-schedulers independently use this algorithm, Nash equilibrium is achieved.

To validate the proposed algorithm, its performance is studied under different system utilizations, heterogeneities, and sizes. The experiment results indicate that our algorithm outperforms the proportional scheme, and is close to the optimal scheme in terms of overall expected response time. However, our algorithm ensures fairness to all schedulers, which is important under a self-interested environment. Trace simulation is also conducted using a job workload log of the Scalable POWERpallel2 system (SP2) in San Diego Supercomputer Center (SDSC). Our algorithm increases the expected response time by a maximum of 14%, but improves fairness by 12–27% in contrast to Opportunistic Load Balancing (OLB), Minimum Execution Time (MET), Minimum Completion Time (MCT), Switching Algorithm (SA), and k-Percent Best (kPB).

The main contributions of this study are as follows.

- It provides a unified framework, which characterizes resources into three roles and uses protocols to describe various interactions.
- It enhances fairness among self-interested schedulers using Nash equilibrium of a non-cooperative game.
- It proposes a fairness aware algorithm based on reinforcement learning, dynamically adapting to job arrival and execution without any priori knowledge.
- It validates the capability of our algorithm to provide fairness and minimize the expected response time under various system utilizations, heterogeneities, and sizes through synthesized tests. Trace simulation shows improved fairness by approximately 20% traded by at most 14% increase in response time, compared with five typical load balancing algorithms.

The remainder of this paper is organized as follows. Section 2 provides the related work on load balancing. In Section 3, a unified framework is presented to describe large-scale distributed systems. Then, Section 4 focuses on the load balancing problem under a self-interested environment, and defines a non-cooperative game. In Section 5, a fairness aware algorithm based on reinforcement learning is proposed. The performance of this algorithm is evaluated in Sections 6 and 7. Finally, Section 8 concludes this paper.

2. Related work

2.1. Static versus dynamic

Load balancing has been studied for decades. During the early stages, Directed Acyclic Graph (DAG) scheduling [12,13] is been investigated for parallel machines. Resources are dedicated in these parallel systems. Task dependency and execution time on resources are possible to acquire. A scheduling scheme is often determined at compile time. Thus, these algorithms are static. Static scheduling requires a priori knowledge of arrival and execution.

However, job arrival and execution are hard to predict because uncertainties exist in distributed systems [14]. For example, the unstable communication consumption of low-speed networks and fluctuating computational capacity of resources cause uncertain execution time of jobs. Predictions based on historical records [15] or workload modeling [16] are used to estimate the execution time of jobs. But unsatisfactory precision and extra complexity are the drawbacks of these methods. Furthermore, jobs arrival patterns vary from different applications. Size and Computation Communication Ratio (CCR) can hardly be predicted. Therefore, dynamic algorithms are popular for load balancing in distributed systems. A scheduling scheme is determined at running time.

Batch mode, which makes scheduling scheme for a fixed number of jobs, is one category of dynamic scheduling. Min–Min (map jobs with least minimum completion time first), Max–Min (map jobs with the maximal minimum completion time first), and Suffrage (map jobs which suffer the most if not allocated right now) [2,17,18] are three typical batch heuristics. Batch functions like a cache to mitigate the influence of uncertain arrival pattern. These algorithms have to wait until all jobs in the batch have arrived, so they lack real-time capability. By contrast, online mode emerges and jobs are scheduled immediately after they arrive. Five such algorithms are available, namely, OLB, MET, MCT, SA, kPB [3,4,6]. OLB assigns jobs to the earliest idle resource without any consideration about the execution time of the job on the resource. MET assigns jobs to a resource that results in the least execution time for that job, regardless of that machines availability. MCT assigns jobs to the resource yielding the earliest completion time. SA first use the MCT until a threshold of balance is obtained followed by MET which creates the load imbalance by assigning jobs on faster resources. kPB tries to combine the best features of MCT and MET simultaneously instead of cyclic manner of SA. In this method, MCT are applied to only k percentage of best resources. However, These algorithms disregard the influence from subsequent jobs.

Dynamic scheduling is shortsighted and does not consider the subsequent jobs. To achieve a global optimization, scholars proposed new dynamic algorithms to adapt to job arrival and execution processes. The authors of [19] presented a resource planner system that reserves resources for subsequent jobs. The authors of [20] proposed a dynamic and self-adaptive task scheduling scheme based upon application-level and system-level performance prediction. An on-line system for predicting batch-queue delay was proposed by Nurmi et al. [21]. Rao and Huh [22] presented a probabilistic and adaptive job scheduling algorithm using system generated predictions for grid systems.

The algorithm proposed in this paper can be classified into dynamic scheduling. Compared with OLB, MET, MCT, SA, and kPB, it can effectively adapt to the job arrival and execution processes. Different with the approaches in [19–22], it does not depend on any workload prediction models.

2.2. Collaboration of schedulers

Collaboration of numerous resources can dramatically improve the performance of distributed systems through load balancing. If the involved schedulers are cooperative (called homo-schedulers in this study), they cooperate at their best to ensure that the entire system runs at its optimum. Grosu et al. [23,24] studied load balancing for homo-schedulers based on cooperative game theory. The work of [25] introduced a decentralized dynamic scheduling approach called community aware scheduling algorithm (CASA). Meanwhile, [26] focused on optimizing the performance of Infrastructure as a Service (IaaS) using a meta-scheduling paradigm across multiple clouds. These algorithms attempt to improve the overall performance of individual domains.

If the schedulers are self-interested (called heter-schedulers in this study), they make decisions independently to maximize their own profits. Fairness is the key to spontaneous collaboration. Two approaches are most commonly used in literature, namely, distributive fairness and game theory.

In distributive fairness scheduling, a fraction of the resources according to predefined shares are ensured. Distributive fairness is implemented through fair queuing mechanism such as Yet another Fair Queueing (YFQ), Start-time Fair Queuing (SFQ) and Four-tag Start-time Fair Queuing (FSFQ) [27], or their modifications [28]. However, those mechanisms do not describe how shares are defined.

The other approach is to optimize the performance (the utility) of users directly, rather than just the allocated resources. Ref. [29] introduced a marketing mechanism on the concept of biding in economics. Subrata et al. [10] modeled the grid load-balancing problem as a noncooperative game. In Ref. [11], a Nash bargaining solution and Nash equilibrium were adopted for single-class and multi-user jobs, respectively. Ref. [30] considered a multi-organizational system in which each organization contributed processors to the global pool, as well as jobs to be processed on the common resources. In a non-monetary approach, jobs are scheduled to minimize the global performance metric with an additional requirement, i.e., the utility of each player cannot be worse than if the player would act alone. In Refs. [31,32], an auction based method was proposed. This method determines the auction winner by applying the game theory mechanism and by holding a repetitive game with incomplete information in a non-cooperative environment.

Game theory is useful in spontaneously forming a federation [33]. The aforementioned algorithms except [29] are all static and assume that job arrival and service conforms to queueing theory. Actually they may fail under a non-Markovian environment. Pezoa and Hayat studied the performance of non-Markovian arrival and execution [34]. Hence, this study proposes a dynamic algorithm for the collaboration of heter-schedulers, which learns a non-cooperative game online and adapts to the job arrival and execution processes [35].

3. A unified framework

The ultimate scale of a distributed system is humongous, considerably similar to the Internet itself. It crosses organizational and national boundaries. Fig. 1 depicts a large-scale distributed system. Resources are separated by administrative domains. A number of schedulers are spread all over the system. Users submit their jobs to these schedulers, and afterward, the schedulers dispatch them for execution. Resources have three roles.

- PE, which is dedicated for computation.
- Homo-scheduler, which cooperates with other homo-schedulers in the same group to handle incoming jobs.

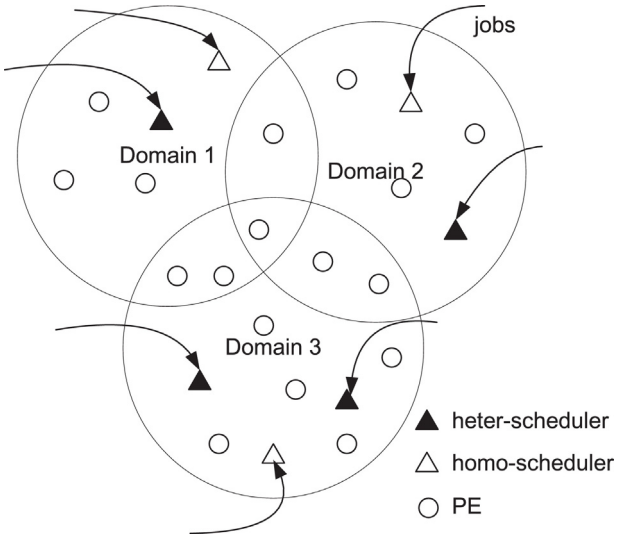


Fig. 1. Overlook of large-scale distributed system.

- Heter-scheduler, which accepts or rejects jobs to maximize its own profit.

Schedulers that share the same interest and fully cooperate with one another to implement a mission at their best, are grouped together. The schedulers in such group are called homo-schedulers. A domain generally forms a group, which is an independent and autonomous entity. Thus, the group behaves individually driven by their own profits. A resource can be a PE, a home-scheduler, or a heter-scheduler.

Fig. 2 shows the structure and the possible relation of roles. PEs are in the charge of one or more schedulers. In the middle layer, the quadrangle represents a domain. Homo-schedulers are grouped together. They dispatch jobs to other schedulers in the same group or to the PEs that are affiliated with it. Each group delegates a heter-scheduler to link to heter-schedulers from other groups. In fact, a heter-scheduler plays a consubstantial role with a homo-scheduler; thus, jobs accepted by the heter-scheduler can be transferred to a homo-scheduler, and finally to the PEs.

An open question is how such a tremendous distributed computing platform, which is likely to be composed of hundreds of thousands of resources, can be maintained and function properly. We believe that various protocols between roles lead to a loosely coupled system.

- Intra-site allocation, which occurs between PEs and a homo-scheduler. Most of the previous scheduling algorithms belong to this protocol, such as Min–Min, MET, MCT, etc.

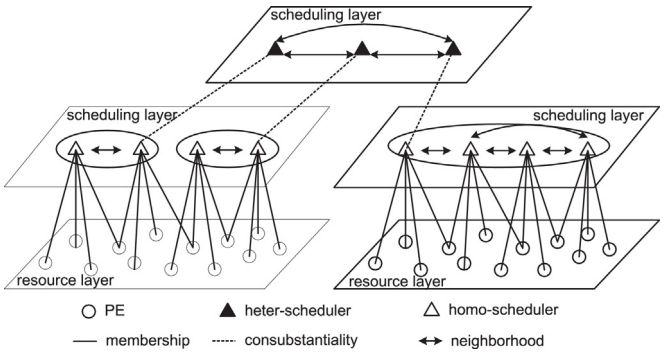


Fig. 2. Structure of large-scale distributed system.

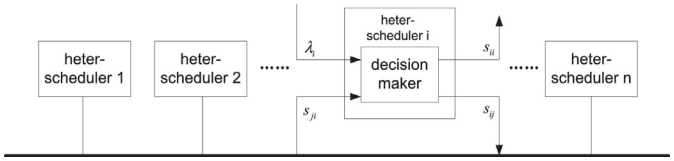


Fig. 3. System model of heter-scheduler collaboration.

- Intra-site cooperation, which occurs among homo-schedulers. The homo-schedulers in a group assist one another to optimize the performance of the entire group. Rescheduling [36] and cooperative game [24] are such protocols.
- Inter-site collaboration, which occurs among heter-schedulers. Schedulers attempt to obtain a joint strategy on behalf of their groups, which results in a win–win situation in terms of their individual profits. This protocol is relatively less researched.

This framework is inspired by the hierarchical semi-selfish Grid model of [8]. However, our version is more general, and the model in [8] becomes one of its special cases. This framework captures the realistic administrative features of a real-life, large-scale distributed computing environment. The protocol concept assists in determining when and where an efficient job scheduling algorithm can be applied.

4. Formalization and modeling

In this study, we focus on the protocol of heter-scheduler collaboration. Suppose n heter-schedulers exist at the top scheduling layer in Fig. 2. Fig. 3 provides the system model of heter-scheduler collaboration.

Jobs arrive at scheduler i with rate λ_i. The scheduler accepts jobs with rate s_{ii} and transfers the remaining jobs to scheduler j with rate s_{ij}. Scheduler i further receives jobs at rate s_{ji} transferred from scheduler j. We have the equations below.

$$\lambda_i + \sum_{j \neq i} s_{ji} = s_{ii} + \sum_{j \neq i} s_{ij} \tag{1}$$

$$\sum_i \lambda_i = \sum_i s_{ii} \tag{2}$$

Let μ_i be the service rate of the group where heter-scheduler i is linked to. If the group is modeled as an M/M/1 queueing system, then Eq. (3) ensures limited queue length.

$$s_{ii} \leq \mu_i \tag{3}$$

The following equation can be inferred from Eqs. (2) and (3).

$$\sum_i \lambda_i \leq \sum_i \mu_i \tag{4}$$

In this model, heter-schedulers decide the admission rate s_{ii} and rejection rate s_{ij} of jobs on their own. The strategy of scheduler i can be represented by the vector $s_i \triangleq \{s_{i1}, s_{i2}, \ldots, s_{ii}, \ldots, s_{in}\}$, and $s \triangleq \{s_1, \ldots, s_n\}$ is the joint strategy.

As stated in Section 3, heter-schedulers have their own interests. They behave independently, and tend to approach their goals. For most applications, response time is of high concern. Thus, each heter-scheduler is assumed to aim at minimizing the response time of the jobs on them. The expected response time RT_i is given by Eq. (5).

$$RT_i(s) = CT_i(s) + TT_i(s) \tag{5}$$

where $CT_i(s)$ is the function of the expected completion time of accepted jobs on scheduler i, and $TT_i(s)$ is the function of the expected transfer time of rejected jobs.

If the M/M/1 model is applicable, then the expected completion time can be computed using Eq. (6).

$$CT_i(s) = \frac{1}{\mu_i - s_{ii}} \tag{6}$$

According to [11], an approximation of the expected transfer time of a job is given by Eq. (7), when the communication network is modeled as an M/M/1 queuing system. Parameter t in Eq. (7) depends on average job size, bandwidth, etc.

$$TT_i(s) = \frac{t}{1 - t\sum_{i=1}^{n}\sum_{\substack{j=1 \\ j \neq i}}^{n} s_{ij}}, \quad \text{where} \quad \sum_{i=1}^{n}\sum_{\substack{j=1 \\ j \neq i}}^{n} s_{ij} < \frac{1}{t} \tag{7}$$

where $\sum_{i=1}^{n}\sum_{\substack{j=1 \\ j \neq i}}^{n} s_{ij}$ is the total traffic through the network.

From Eqs. (5)–(7), the decision of a scheduler is influenced by the scheduling strategies of other schedulers. Once the schedulers become self-interested, the scheduling scenario becomes similar to a game. After the competition, a state is reached wherein none of the schedulers wants to change their strategies. That is, no schedulers can further minimize their response time by unilaterally adjusting their strategies. This state is called Nash equilibrium in game theory. In fact, the collaboration of heter-schedulers aims to find a joint strategy, which leads to Nash equilibrium.

According to game theory, we use the following game to define our load balancing problem.

Definition 4.1 *((The load balancing game of heter-schedulers)).* The load balancing game of heter-schedulers consists of:

— Players: n heter-schedulers.

— Strategies: the joint strategy $s \triangleq \{s_1, \ldots, s_n\}$, which consists of the admission and rejection rates of each scheduler.

— Preference: the preference of each player is represented by its expected response time $RT_i(s)$. Each player i prefers the strategy s to the strategy s' if and only if $RT_i(s) < RT_i(s')$.

A unique Nash equilibrium for the game exists because the expected response time functions (see Eqs. (5)–(7)) are continuous, convex, and increasing [37]. Refs. [10,11] proposed two static algorithms to achieve the equilibrium of the aforementioned game based on queueing theory. However, they require predicting the parameters, such as the arrival rate, service rate, and so on. Furthermore, these algorithms may fail under a non-Markovian environment. Thus, in the next section, we propose an online learning based algorithm, which no longer requires predicting parameters or using Eqs. (6) and (7) that are constrained by the Markovian environment.

5. Fairness aware adaptable load balancing algorithm

We must solve the aforementioned game for our load balancing scheme. A solution for this problem is the joint strategy at Nash equilibrium, which is subject to the constraints Eqs. (1)–(3). The definition is provided as follows.

Definition 5.1 *((Nash equilibrium)).* A Nash equilibrium of the load balancing game defined in the preceding section is a joint strategy s, such that for every scheduler $i (i = 1, \ldots, n)$:

$$s_i \in \underset{\tilde{s}_i}{\arg\min} RT_i(s_1, \ldots, \tilde{s}_i, \ldots, s_n)$$

We use a typical problem in game theory, namely, the prisoners' dilemma (see Fig. 4), to demonstrate the motivation of our method.

		B: confess	B: deny
A	confess	(-2,-2)	(0,-5)
A	deny	(-5,0)	(-1,-1)

Fig. 4. Prisoners' dilemma.

Two prisoners, A and B, both have two choices, i.e. to confess or deny. The numbers from left to right within the parentheses denote the years spent in jail of A and B. If A confesses and B denies, A will be set free, whereas B will be prisoned for 5 years. Obviously, B has the incentive to confess. When both choose to confess, neither prefers to deny. Thus, the strategy (confess, confess) is a Nash equilibrium. However, the strategy (deny, deny) is optimal (called the Pareto Optimality) rather than (confess, confess). This strategy can only be realized through the cooperation between A and B; otherwise, both have the incentive to alter their choices. By analogy, Definition 5.1 appears to indicate that each scheduler attempts to minimize its own response time, but ensures a balanced load among all schedulers, because no one will profit form the deviation in the current strategy. That is, if the strategy is unfair, the inferior scheduler will deviate from it. In summary, the expected response time decreases as much as possible while fairness is ensured. The rest of this section describes the process of finding Nash equilibrium.

Before providing the algorithm, we transform the strategy of a scheduler into probabilities for convenience, to satisfy the constraints Eqs. (1)–(3) completely. The strategy is redefined as the probability that the scheduler will allocate jobs to other schedulers. $\beta_i = \{\beta_{i1}, \ldots, \beta_{in}\}(\sum_{j=1}^{n}\beta_{ij} = 1)$ denotes this new strategy, where β_{ij} is the probability that scheduler i will dispatch jobs to scheduler j. The strategies β_i and s_i are equivalent. The following relation holds between admission or rejection rate s_{ij} and probability β_{ij}.

$$s_{ij} = \beta_{ij} \cdot [\lambda_i + \sum_j s_{ji}] \tag{8}$$

Given that $\sum_{j=1}^{n}\beta_{ij} = 1$, Eq. (1) is satisfied.

When solving the problem presented in Definitions 4.1 and 5.1, each scheduler requires knowledge on the job arrival and execution processes, the strategies of other schedulers, network performance, etc. This creates an extremely complicated problem. In this study, we use the machine learning approach to learn the aforementioned non-cooperative game. Machine learning is a discipline that researches approaches for mining knowledge automatically through samples only.

Reinforcement learning is an online approach, which strengthens the strategies with better performance. The samples are the historical allocations. The distinguished advantage of this approach is that it is model-free, and thus, it does not require obtaining background information, such as job arrival process or network performance. In this study, we adopt Q-learning, which is the most frequently used reinforcement learning algorithm [38].

The Q-function in Q-learning represents the accumulative response time. Each time a job is allocated, we track its response time. And then update its Q-function. Eq. (9) shows the update of scheduler i when a job is allocated to j.

$$Q_i(Agt_j) = (1-\alpha)Q_i(Agt_j) + \alpha[r + \eta \max_k Q_i(Agt_k)] \tag{9}$$

where Agt_j denotes the scheduler j, $\alpha \in (0, 1]$ is the learning rate, r is a monotone decreasing function of the real-time response time, and η is the discounted factor which is a parameter in Q-learning.

α is iterated as follows. Eq. (10) ensures the convergence of Eq. (9).

$$\alpha = \frac{1}{1 + visits(Agt_j)} \tag{10}$$

where $visits(Agt_j)$ is the number of allocation to Agt_j so far.

The following equation provides the strategy β_i of scheduler i.

$$\beta_{ij} = \frac{Q_i(Agt_j)}{\sum_k Q_i(Agt_k)} \tag{11}$$

Eqs. (9)–(11) provides the algorithm for a scheduler to find the strategy with the least response time. However, two additional problems exist. First, achieving equilibrium is difficult for the schedulers when this algorithm is used. The strategies of each scheduler are oscillating. Thus, the mechanisms have to be in place to determine whether an equilibrium is reached and to implement strategy adjustment.

When an equilibrium is reached, none of the schedulers is willing to violate its current strategies. After scheduler i attempts to change from its old strategy β_i^{old} to the new strategy β_i^{new}, the performance of other schedulers are observed. If m schedulers exist, the performance of which improves by adjusting their strategies, then it is believed that m schedulers benefit from the violation. The strategy of scheduler i is updated according to the probability in the way below.

$$\beta_i \overset{\frac{m}{n}}{\leftarrow} \beta_i^{old} \quad \text{or} \quad \beta_i \overset{1-\frac{m}{n}}{\leftarrow} \beta_i^{new} \tag{12}$$

In the equation above, if $m = 0$, no schedulers benefit from strategy violation and β_i^{new} is accepted with probability one. if $m = n$, a considerable deviation from the equilibrium occurs and β_i^{old} is maintained.

When all the schedulers have stopped updating, a joint strategy at the Nash equilibrium is obtained.

The second problem is job forwarding loop. There exists a probability, though it is small, that a job will be forwarded all the time. To avoid this phenomenon, a job is only allowed to be forwarded by finite times before being admitted by a scheduler. Multiple forwarding is unexpected because it increases the response time of jobs and wastes network bandwidth.

In the synthesized tests, it is observed that most jobs are admitted in three hops. Considering the limit of forwarding no more than three times, all the incoming jobs are admitted in the end. Consequently, the constraint of Eq. (2) is satisfied.

The fairness aware adaptable load balancing algorithm for a single scheduler, which is named Fairness aware Adaptable Scheme (FAS), is given below. Fig. 5 shows the flowchart of the FAS algorithm. This algorithm is distributed. Each scheduler runs this algorithm locally and stops at the strategy of the unique equilibrium.

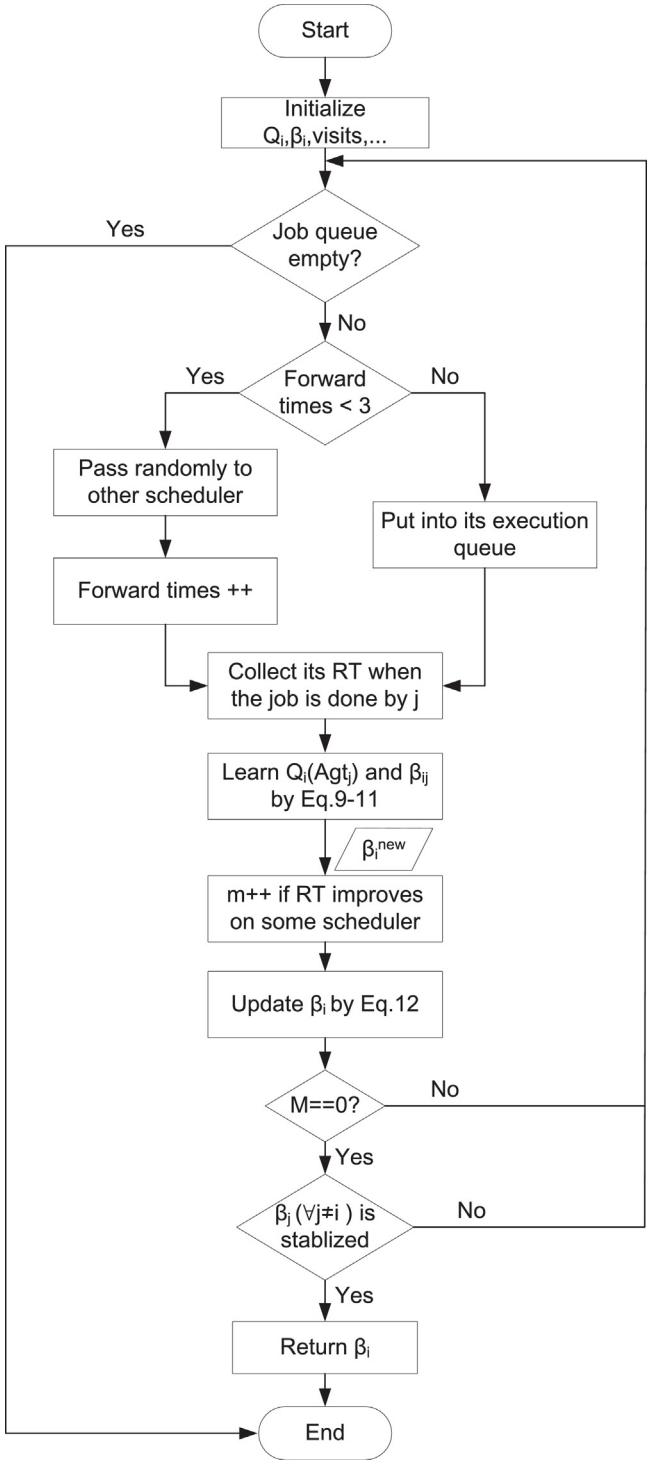


Fig. 5. Flowchart for FAS algorithm.

Algorithm. **FAS**: Fairness aware Adaptable Scheme for Scheduler i

Input:

Set $\beta_{ij} = \frac{1}{n}, Q_i(j) = 0, 1 < j < n$;

Output:

strategy β_i at Nash equilibrium;

while job queue not empty **do**
  if Forward Times = 3 **then**
    Allocate jobs to itself;
  else
    Allocate jobs randomly;
    Increase its forward times by 1;
  end if
  if Current job is assigned to scheduler j **then**
    Collect its response time r from j;
    Compute new strategy β_{ij}^{new}:
    $Q_i(Agt_j) = (1-\alpha)Q_i(Agt_j) + \alpha[r + \eta \max_k Q_i(Agt_k)]$
    $\beta_{ij}^{new} = \frac{Q_i(Agt_j)}{\sum_k Q_i(Agt_k)}$
    Count the number m of better schedulers;
    Update its strategy by $\beta_i \overset{\frac{m}{n}}{\leftarrow} \beta_i^{old}$ or $\beta_i \overset{1-\frac{m}{n}}{\leftarrow} \beta_i^{new}$;
    if $m = 0$ **then**
      Inform other schedulers this possible equilibrium strategy;
      if All the schedulers report their equilibrium strategy **then**
        return β_i;
      end if
    end if
  end if
end while

6. Synthesized tests

We perform simulations to study the impact of system utilization, heterogeneity, and size, on the performance of our scheme. The system parameters that are used in our experiments are similar to those in [11]. There are 32 fully connected candidate schedulers. Table 1 provides the system configuration. The job arrival rate of scheduler i can be computed using the arrival fraction q_i. $\lambda_i = q_i \times \lambda$ where λ is the overall arrival rate $\sum_i \lambda_i$. The mean communication time t is assumed to be 0.001 s. Transferring a job takes time t on average.

The performance metrics used are the expected response time at the equilibrium and the fairness index. The latter is important among self-interested schedulers and is defined as follows,

$$I(C) = \frac{\left[\sum_{j=1}^{n} C_j\right]^2}{n\sum_{j=1}^{n} C_j^2} \tag{13}$$

Input C is the vector $C = (C_1, C_2, \ldots, C_n)$ where C_j is the expected response time on scheduler j. If $I = 1$, all the schedulers have the same expected response time, which indicates a balanced load. If $I < 1$, some schedulers are preferred and undertake more load.

For comparison, the following two load balancing schemes are implemented:

– Global Optimal Scheme (GOS) [39]: This scheme minimizes the expected response time across all the jobs executed by the system, to provide a systematically optimal solution. The loads for each schedulers are obtained by solving the following non-linear optimization problem:

$$\min \overline{RT} = \frac{1}{\lambda}\sum_i \left[\sum_j s_{ij} CT_j + (\lambda_i - s_{ii})TT_i\right] \tag{14}$$

Table 1
System configuration I.

Schedulers i	Service rate μ_i (jobs/s)	Arrival fraction q_i[a] ($q_i \in [0, 1]$)
1–6[b]	10	0.0025
7–12[b]	50	0.01
13	100	0.02
14–18[b]	100	0.025
19–22[b]	150	0.04
23–26[b]	200	0.05
27–32[b]	250	0.07

[a] $\sum_i q_i = 1$.
[b] Identical configurations for schedulers among it.

– Proportional Scheme (PROP_M) [40]: According to this scheme, each scheduler allocates its jobs to itself or to others in proportion to their processing rate as follows:

$$s_{ij} = \lambda_i \times \frac{\mu_j}{\sum_k \mu_k} \tag{15}$$

Although GOS can provide the optimal solution, it exhibits some impractical constraints. First, it assumes that the node and the network can be modeled by M/M/1. However, Poisson distribution may not be the case in the job arrival and transfer processes. Second, GOS is a static algorithm. It is a nontrivial job to obtain such input parameters as the arrival rate, service rate, and network constant, etc. Third, GOS assumes that other schedulers are cooperative. More jobs are assigned to the nodes of higher performance and fairness is lost. Hence, GOS is not competent for load balancing among self-interested schedulers.

We present and discuss the simulation results in the following sections.

6.1. Convergence of best-response strategy

The algorithm runs where the initial strategy s_i of each player i is the vector of element $1/n$. Each player then refines and updates its strategy at each scheduling run. In the first set of experiments, we study the convergence to the best-response strategy, which is the best strategy while others keep their strategies static. We simulated a heterogeneous system of 32 schedulers and the average system utilization (the definition of which is provided in Section 6.2) is set to 60%, i.e. the overall arrival rate λ is 2316. The strategies of schedulers are initially generated randomly subject to the constraints Eqs. (1)–(3). One scheduler is allowed to run the algorithm while the others keep their initial strategies.

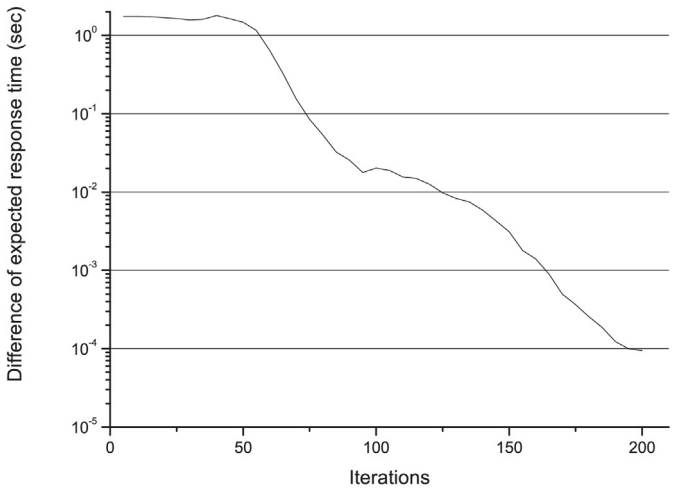
The experiment above is repeated 20 times. The strategies of the schedulers are randomly initialized each time. The expected response time on average of the 20 experiments is observed. Fig. 6(a) shows the difference of the expected response time from the last strategy on a certain scheduler when the system utilization is 60%. The absolute difference decreases to 10^{-4} after about 200 iterations, which is considered converged. Besides, Fig. 6(b) provides the standard deviation of the expected response time over 20 repetitions at each iteration. In the end, the expected response time is around 0.01 s with standard deviation 0.002. After 110 iterations, the standard deviation varies less and the decrease in the expected response time mainly results from the strategy improvement.

Fig. 7 shows the number of iterations for convergence under different system utilizations. As system utilization increases, our algorithm requires more iterations to obtain the best-response strategy. The system will not reach a steady state until a larger quantity of jobs has arrived. For a heavy utilization rate of 90%, convergence rate increases to around 460 iterations.

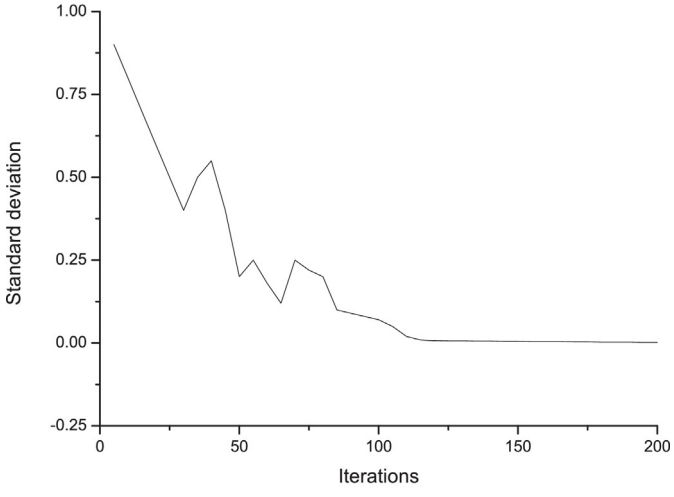
Fig. 7 also displays the convergence speeds of algorithms from [10,11]. The convergence speed of algorithm [11] keeps unchanged with the system utilization because it is not a random search algorithm. Its time complexity only depends on the system scale. The convergence speed of our algorithm is higher and higher than that of algorithms [10] as system utilization increases, because more time is needed to spend on learning the job arrival and execution processes. But we believe a slower speed is tolerable, because only hundreds of jobs, out of thousands processed by a large-scale distributed system, are influenced before convergence.

6.2. Effect of system utilization

We simulated a heterogeneous system that consists of 32 schedulers (all the candidates in Table 1) to study the effect of system



(a) Expectation



(b) Standard deviation

Fig. 6. Convergence to best-response strategy.

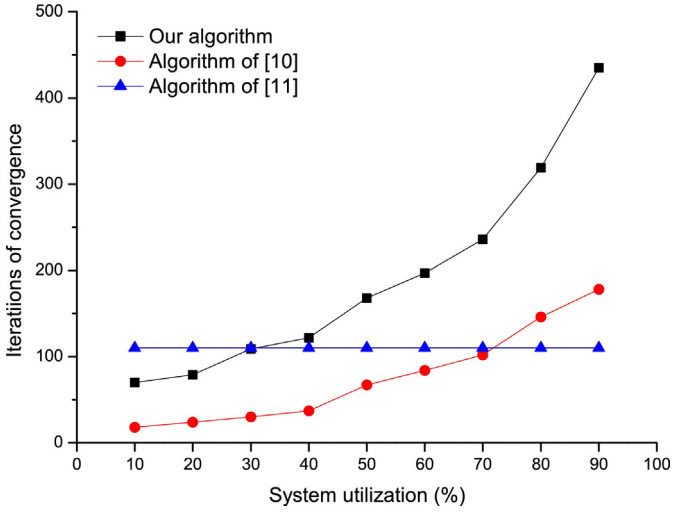
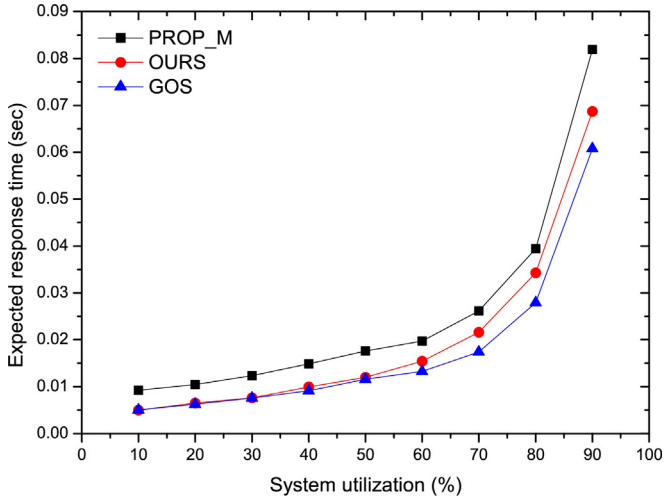
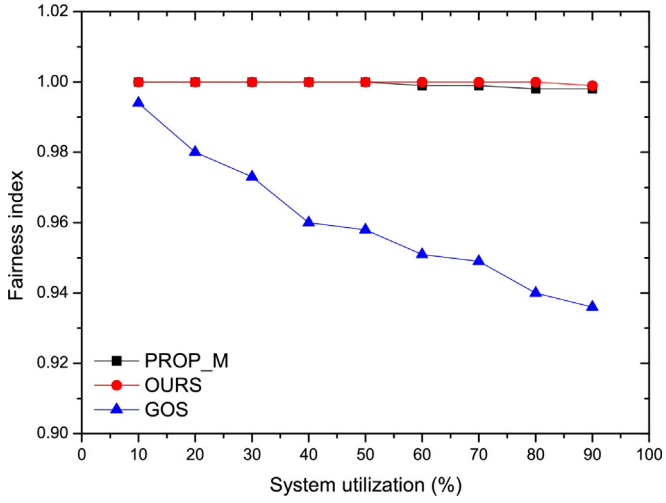


Fig. 7. Number of iterations versus system utilization.



(a) Expected response time



(b) Fairness index

Fig. 8. Effect of system utilization.

utilization. We use ρ to symbolize system utilization, which is computed by the formula $\lambda/\sum_i \mu_i$ and satisfies $0 < \rho \leq 1$. As ρ increases, system load becomes heavier.

The results, with ρ ranging from 10% to 90%, are presented in Fig. 8. Fig. 8(a) shows the expected response times under different system utilizations. As system utilization increases, the expected response time becomes longer. The algorithm of PROP_M exhibit the worst performance because it significantly overloads the less powerful schedulers. The algorithm GOS exhibits the best performance because it provides a systematically optimal solution. Our algorithm exhibits sub-optimal performance. For example, at 60% system utilization, the response time is around 25% less than that of PROP_M and around 10% greater than that of GOS. Decentralization and rationality are the main advantages of non-cooperative game.

Fig. 8(b) presents the fairness indices of the three algorithms. The fairness indices of our algorithm and PROP_M are around 1. They behave fairly to all schedulers. The fairness index of GOS ranges from around 1 to 0.935 at a heavy load. Fig. 8 shows the feature of heter-schedulers, that is, absence of full cooperation because they are self-interested.

To gain insight into the results shown in Fig. 8, Fig. 9 presents the expected response time of each scheduler at 60% system utilization. The 32 schedulers nearly have identical expected response time under our algorithm and PROP_M. GOS is unfair, with varying

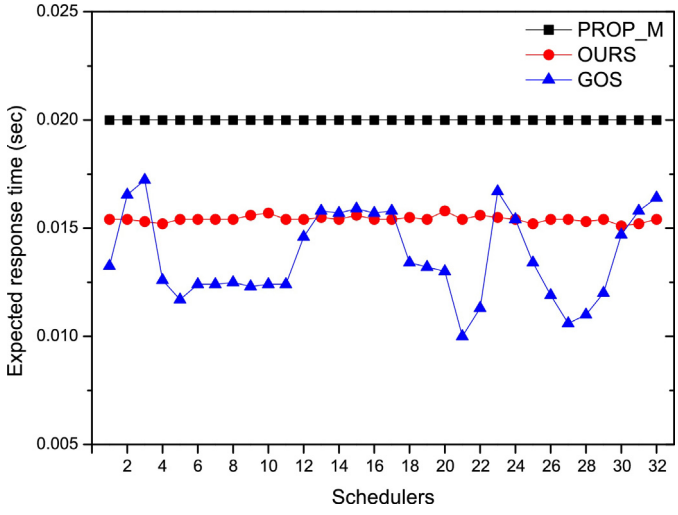


Fig. 9. Expected response time of each scheduler ($\rho = 60\%$).

expected response times on schedulers. GOS decreases the overall expected response time through cooperation.

6.3. Effect of heterogeneity

In this section, we study the effect of heterogeneity on the performance of load balancing schemes. A simple means to characterize system heterogeneity is to use the processor speed. One of the common measures for heterogeneity is speed skewness which is defined as the ratio of the maximum processing rate to minimum processing rate of the computers in the system. We investigate the effectiveness of load balancing schemes by varying speed skewness.

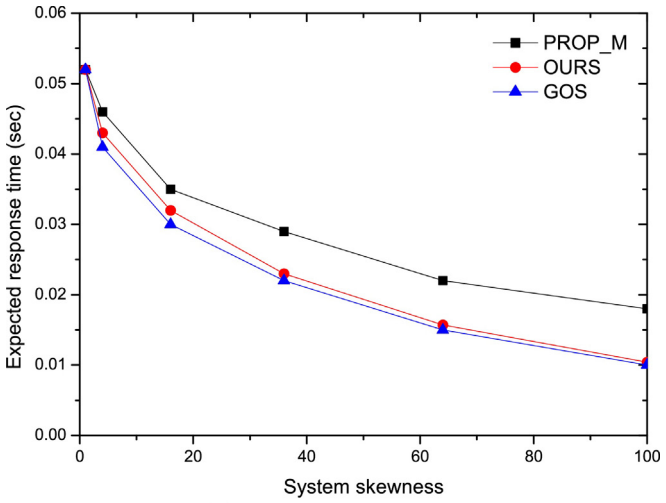
A 32-scheduler system is simulated, but the schedulers are divided into three groups as shown in Table 2. Schedulers 1 to 8 represent the fast group. Schedulers 9 to 24 represent the moderate group. Schedulers 25 to 32 represent the slow group. A homogeneous system is simulated at the beginning. Six experiments are conducted by varying speed skewness. System utilization is 60% in these experiments. The overall arrival rate is also provided in Table 2, and the job arrival fraction q_i is found in Table 1.

The results are presented in Fig. 10. As speed skewness increases, the computational power of the system also increases. Thus, the expected response time decreases. For a homogeneous system, the three algorithms exhibit the same performance. When the system becomes more heterogeneous, our algorithm and GOS decrease the expected response time faster. When speed skewness is greater than 50, our algorithm is close to GOS, which indicates our algorithm is very effective in highly heterogeneous systems.

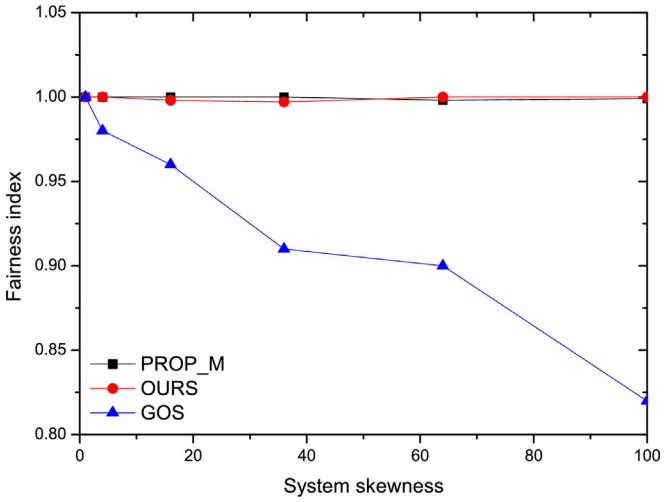
For the fairness index in Fig. 10(b), our algorithm and PROP_M maintain a fairness index of nearly 1 with increasing speed skewness. The fairness index of GOS decreases from 1 at low skewness to 0.82 at high skewness. GOS produces an allocation that does not guarantee equal expected response times, particularly, at high skewness. A balanced load and near-optimal performance are the main advantages of our algorithm.

Table 2
System configuration II.

Speed skewness	1	4	16	36	64	100
$\mu_i (i = 1:8)$	10	100	200	300	400	500
$\mu_i (i = 9:24)$	10	50	50	50	50	50
$\mu_i (i = 25:32)$	10	25	12.5	8.33	6.25	5
Overall arrival rate	192	1080	1500	1960	2430	2904



(a) Expected response time



(b) Fairness index

Fig. 10. Effect of system heterogeneity.

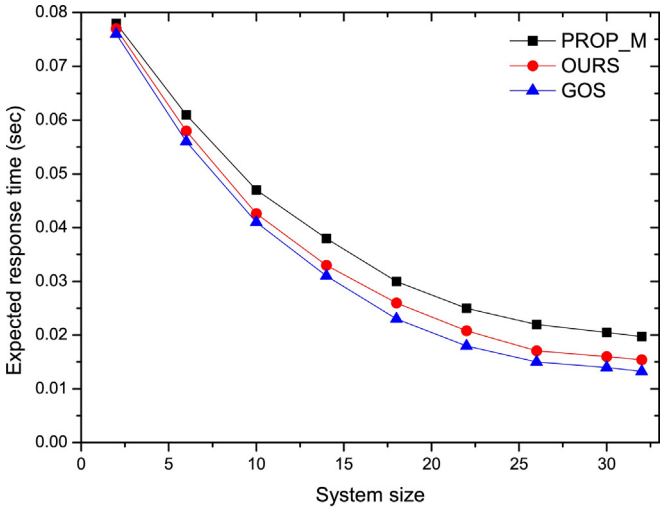
6.4. Effect of system size

In this group of experiments, we vary the number of schedulers available in the system and investigate the effect of system size. We use the schedulers listed in Table 1 and vary the number of schedulers in the system from 2 to 32. The system initially includes two schedulers, namely, the fastest and the slowest. System size increases by four at each experiment. A half of the increments are the fastest picked up from the rest, while the other half are the slowest. For instance, when the system size is 14 nodes, it includes 6 nodes of service rate 10, 6 of 250, 1 of 50, and 1 of 200. In this manner, speed skewness remains 25 in this group of experiments. During each experiment, system utilization remains 60% across all the experiments. Table 3 shows the number of schedulers at each service rate and the overall arrival rate for each system size.

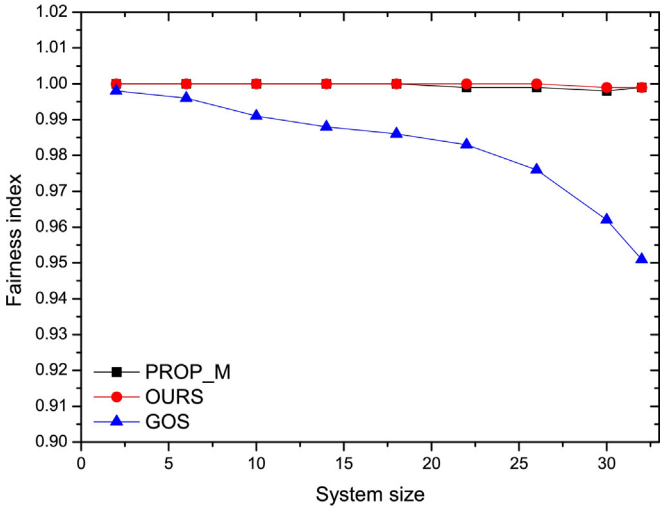
Fig. 11 compares the three algorithms in terms of the expected response time and fairness index. In Fig. 11(a), the expected response time decreases as the system size increases from 2 to 32. The performance of our algorithm lies between those of GOS and PROP_M. The overall expected response time is considerably less in the case of our algorithm compared with PROP_M. As shown in Fig. 11(b), the fairness indices of our algorithm and PROP_M are very close to 1 with an increase in system size. However, the fairness index of GOS considerably declines with an increase in system size.

Table 3
System configuration III.

Size	λ	μ_i (jobs/s)					
		10	50	100	150	200	250
2	156	1	0	0	0	0	1
6	468	3	0	0	0	0	3
10	780	5	0	0	0	0	5
14	1086	6	1	0	0	1	6
18	1386	6	3	0	0	3	6
22	1656	6	5	0	1	4	6
26	1926	6	6	1	3	4	6
30	2196	6	6	4	4	4	6
32	2316	6	6	6	4	4	6



(a) Expected response time



(b) Fairness index

Fig. 11. Effect of system size.

The expected response time of each scheduler varies. Some schedulers are preferred. Fig. 11 validates that our algorithm balances load and performs near the optimum.

7. Trace simulation

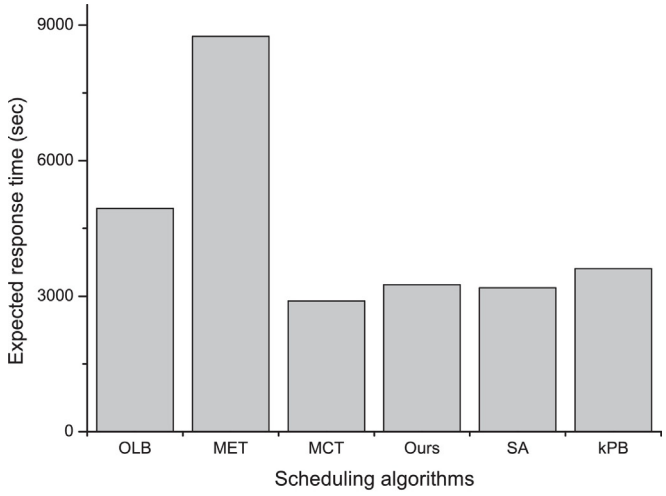
In this section, we will validate the fairness scheme under the track SP2 job workload log of SDSC [41]. There are 128 nodes. 7 queues accommodate the arriving jobs. Each node corresponds

to a heter-scheduler and only 7 heter-schedulers admit coming jobs. Suppose the 128 heter-schedulers are fully connected peers. A scheduler decides whether to accept or reject jobs based on its own performance.

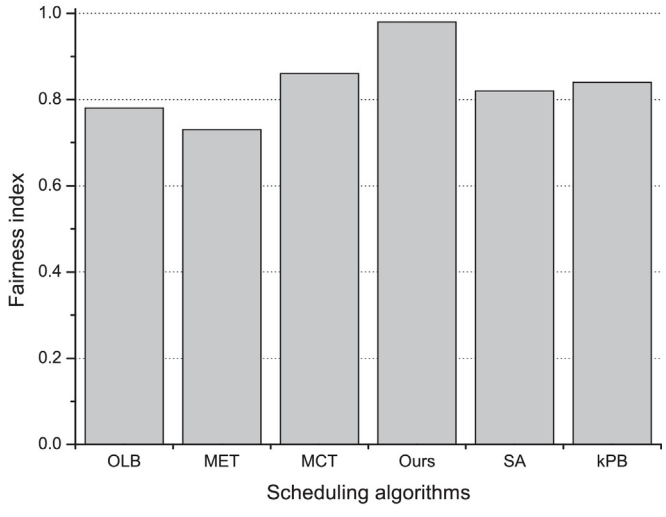
The jobs in the log are assumed to be atomic. According to the log, the arrival times of all jobs are provided. Thus, a real job arrival process is simulated in these experiments. A total of 73,496 records exist, which are divided into 3 folds for k-fold cross validation. The expected response time is averaged over 3 experiments in which one fold is selected as the test data set while the remaining 2 folds are used as the training data set. The execution costs of each job on the 128 PEs are generated by a random number generator, which follows an exponential distribution. The actual run time provided in the workload log is used as the expectation of the random number generator.

Jobs arrive based on the time interval in the log. The initial strategy is stochastic during the learning process. Then the strategy is updated according to Eqs. (9)–(12) once a job is assigned. After learning converges or terminates, jobs are assigned according to the final strategy.

In Section 2.2, the collaboration algorithms among heter-schedulers are static. Parameters such as arrival rate and service rate are prior knowledge, which are difficult to obtain from the SP2 workload log. Therefore, five immediate mode scheduling algorithms, namely, OLB, MET, MCT, SA, and kPB, are selected to be compared with our proposed method.



(a) Expected response time



(b) Fairness index

Fig. 12. SP2 trace simulation.

Fig. 12(a) provides the expected response time of the six algorithms while Fig. 12(b) displays their fairness indices. MCT assigns jobs to the node that yields the earliest completion time, which results in the minimum expected response time. MET assigns jobs to the node with the least execution time, which overloads the fastest node and leads to load imbalance. The fairness index of MET is about 0.7. OLB assigns jobs to the earliest idle node. SA and kPB attempt to combine the best features of MCT and MET. The performance of these algorithms is between those of MCT and MET. Compared with these five algorithms, our algorithm obtains a considerably higher fairness index, although its expected response time is reasonably lower than the best. The compared five algorithms never consider the performance of each individual node. They assume that other schedulers are cooperative and assign jobs to the best of their knowledge. Consequently, the expected response time is improved. However, the fairness, which is important to self-interested schedulers, cannot be ensured.

8. Conclusion

New computation forms, such as Grid and Cloud computing, indicate that large-scale distributed computing plays an increasingly important role in our daily life. Unlike in parallel or small-scale distributed environments, resources are gathered from multiple domains in those systems. Some resources tend to be cooperative, and some are self-interested. Thus, the scheduling algorithms are supposed to differ for various resources. With this consideration, we first build a unified framework for large-scale distributed systems in this study. The resources are classified into PE, homo-schedulers, and heter-schedulers. The different scheduling algorithms can be used as protocols among various classes of resources.

We focus on the scheduling problem for heter-schedulers and modeled it as a non-cooperative game. Then we propose an online scheduling algorithm based on reinforcement learning. This algorithm requires significantly less system knowledge than static algorithms. Through independent learning, the strategies of the schedulers achieve Nash equilibrium. The simulation experiments demonstrate that our algorithm exhibits good performance in terms of the expected response time and fairness.

Acknowledgements

We are very grateful to anonymous reviewers for the comments and suggestions which greatly improved the quality of the manuscript. This research was partially funded by the National Natural Science Foundation of China (Grant Nos. 61402157 and 61502165), and supported by the Hunan Provincial Natural Science Foundation (Grant No. 13JJ4038).

References

[1] M. Armbrust, A. Fox, R. Griffith, A.D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, M. Zaharia, A view of cloud computing, Commun. ACM 53 (4) (2010) 50–58.
[2] M. Maheswaran, S. Ali, H. Siegel, D. Hensgen, R. Freund, Dynamic mapping of a class of independent tasks onto heterogeneous computing systems, J. Parallel Distrib. Comput. 59 (2) (1999) 107–131.
[3] F. Xhafa, L. Barolli, A. Durresi, Immediate mode scheduling of independent jobs in computational grids, in: Proc. the 21st International Conference on Advanced Networking and Applications, 2007, pp. 970–977.
[4] T. Braun, A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems, J. Parallel Distrib. Comput. 61 (6) (2001) 810–837.
[5] G. Khanna, U. Catalyurek, T. Kurc, P. Sadayappan, J. Saltz, A data locality aware online scheduling approach for I/O-intensive jobs with file sharing, in: Proc.

the 12th International Conference on Job Scheduling Strategies for Parallel Processing, JSSPP'06, Springer, 2006, pp. 141–160.
[6] R. Sahu, A. Chaturvedi, Many-objective comparison of twelve grid scheduling heuristics, Int. J. Comput. Appl. 13 (6) (2011) 9–17.
[7] I. Rao, E.-N. Huh, S. Lee, T. Chung, Distributed, scalable and reconfigurable inter-grid resource sharing framework, in: Proceedings of the 2006 International Conference on Computational Science and Its Applications – Volume Part II, ICCSA '06, 2006, pp. 390–399.
[8] Y.-K. Kwok, K. Hwang, S. Song, Selfish grids: game-theoretic modeling and NAS/PSA benchmark evaluation, IEEE Trans. Parallel Distrib. Syst. 18 (5) (2007) 621–636.
[9] D. Grosu, A.T. Chronopoulos, Noncooperative load balancing in distributed systems, J. Parallel Distrib. Comput. 65 (9) (2005) 1022–1034.
[10] R. Subrata, A. Zomaya, B. Landfeldt, Game-theoretic approach for load balancing in computational grids, IEEE Trans. Parallel Distrib. Syst. 19 (1) (2008) 66–76.
[11] S. Penmatsa, A.T. Chronopoulos, Game-theoretic static load balancing for distributed systems, J. Parallel Distrib. Comput. 71 (4) (2011) 537–555.
[12] Y.-K. Kwok, I. Ahmad, Dynamic critical-path scheduling: an effective technique for allocating task graphs to multiprocessors, IEEE Trans. Parallel Distrib. Syst. 7 (5) (1996) 506–521.
[13] Y.-K. Kwok, I. Ahmad, Static scheduling algorithms for allocating directed task graphs to multiprocessors, ACM Comput. Surv. 31 (4) (1999) 406–471.
[14] L.D. Dhinesh Babu, P. Venkata Krishna, Honey bee behavior inspired load balancing of tasks in cloud computing environments, Appl. Soft Comput. 13 (5) (2013) 2292–2303.
[15] L. Yang, J.M. Schopf, I. Foster, Conservative scheduling: using predicted variance to improve scheduling decisions in dynamic environments, in: Proceedings of the 2003 ACM/IEEE conference on Supercomputing, SC'03, 2003, pp. 31–41.
[16] L. Gong, X.-H. Sun, E.F. Watson, Performance modeling and prediction of nondedicated network computing, IEEE Trans. Comput. 51 (9) (2002) 1041–1055.
[17] K. Etminani, M. Naghibzadeh, A min–min max–min selective algorithm for grid task scheduling, in: IEEE/IFIP International Conference in Central Asia on Internet, 2007, pp. 1–7.
[18] S.S. Chauhan, R.C. Joshi, A weighted mean time min–min max–min selective scheduling strategy for independent tasks on grid, in: IEEE International Advance Computing Conference, IACC'10, 2010, pp. 4–9.
[19] S. Jang, X. Wu, V. Taylor, Using performance prediction to allocate grid resources, Tech. rep., GriPhyN, 2004.
[20] X. Sun, M. Wu, Grid harvest service: a system for long-term, application-level task scheduling, in: Proc. the 17th International Symposium on Parallel and Distributed Processing, IPDPS'03, 2003, pp. 25–33.
[21] D. Nurmi, J. Brevik, R. Wolski, Qbets: Queue bounds estimation from time series, in: 13th International Workshop Job Scheduling Strategies for Parallel Processing, JSSPP'07, Springer Berlin Heidelberg, 2007, pp. 76–101.
[22] I. Rao, E.-N. Huh, A probabilistic and adaptive scheduling algorithm using system-generated predictions for inter-grid resource sharing, J. Supercomput. 45 (2) (2008) 185–204.
[23] D. Grosu, A.T. Chronopoulos, M.-Y. Leung, Load balancing in distributed systems: an approach using cooperative games, in: Proceedings of the 16th International Parallel and Distributed Processing Symposium, IPDPS'02, 2002, pp. 196–205.
[24] D. Grosu, A.T. Chronopoulos, M.Y. Leung, Cooperative load balancing in distributed systems, Concurr. Comput. Pract. Exp. 20 (16) (2008) 1953–1976.
[25] Y. Huang, N. Bessis, P. Norrington, P. Kuonen, B. Hirsbrunner, Exploring decentralized dynamic scheduling for grids and clouds using the community-aware scheduling algorithm, Future Gener. Comput. Syst. 29 (1) (2013) 402–415.
[26] S. Sotiriadis, N. Bessis, A. Anjum, R. Buyya, An inter-cloud meta-scheduling (ICMS) simulation framework: architecture and evaluation, IEEE Trans. Serv. Comput. PP (99) (2015) 1–13.
[27] W. Jin, J.S. Chase, J. Kaur, Interposed proportional sharing for a storage service utility, SIGMETRICS Perform. Eval. Rev. 32 (1) (2004) 37–48.
[28] A. Wierman, Fairness and Scheduling in Single Server Queues, 2011.
[29] H. Hanna, A.-I. Mouaddib, Task selection problem under uncertainty as decision-making, in: Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems: Part 3, AAMAS'02, 2002, pp. 1303–1308.
[30] P. Skowron, K. Rzadca, Non-monetary fair scheduling: a cooperative game theory approach, in: Proceedings of the Twenty-fifth Annual ACM Symposium on Parallelism in Algorithms and Architectures, SPAA'13, 2013, pp. 288–297.
[31] H. Wang, H. Tianfield, Q. Mair, Auction based resource allocation in cloud computing, Multiagent Grid Syst. 10 (1) (2014) 51–66.
[32] A. Nezarat, G. dastghaibifard, Efficient Nash equilibrium resource allocation based on game theory mechanism in cloud computing by using auction, in: Proceedings of International Conference on Next Generation Computing Technologies, NGCT'15, 2015, pp. 1–5.
[33] R. Buyya, R. Ranjan, R.N. Calheiros, Intercloud: utility-oriented federation of cloud computing environments for scaling of application services, in: Proceedings of the 10th International Conference on Algorithms and Architectures for Parallel Processing(ICA3PP), 2010, pp. 13–31.
[34] J. Pezoa, M. Hayat, Performance and reliability of non-Markovian heterogeneous distributed computing systems, IEEE Trans. Parallel Distrib. Syst. 23 (7) (2012) 1288–1301.
[35] S.S. Mousavi, B. Ghazanfari, N. Mozayani, M.R. Jahed-Motlagh, Automatic abstraction controller in reinforcement learning agent via automata, Appl. Soft Comput. 25 (2014) 118–128.
[36] R. Sakellariou, H. Zhao, A low-cost rescheduling policy for efficient mapping of workflows on grid systems, Sci. Program. 12 (4) (2004) 253–262.
[37] A. Orda, R. Rom, N. Shimkin, Competitive routing in multiuser communication networks, IEEE/ACM Trans. Netw. 1 (5) (1993) 510–521.
[38] C. Watkins, P. Dayan, Q-learning, Mach. Learn. 8 (1992) 279–292.
[39] K.I.M. Chonggun, H. Kameda, Optimal static load balancing of multi-class jobs in a distributed computer system, in: Proc. of the 10th Intl. Conf. on Distributed Computing Systems, 1990, pp. 562–569.
[40] Y.C. Chow, W.H. Kohler, Models for dynamic load balancing in a heterogeneous multiple processor system, IEEE Trans. Comput. 28 (5) (1979) 354–361.
[41] Anon, http://www.cs.huji.ac.il/labs/parallel/workload/logs.html.