Received: 2 February 2015 | Revised: 22 April 2016 | Accepted: 20 July 2016

DOI 10.1002/cpe.3939

WILEY

RESEARCH ARTICLE

Self-adaptation and mutual adaptation for distributed scheduling in benevolent clouds

Zheng Xiao[1*] | Pijun Liang[1] | Zhao Tong[2] | Kenli Li[1] | Samee U. Khan[3] | Keqin Li[1,4]

[1]College of Information Science and Engineering, Hunan University, Changsha, Hunan, China
[2]College of Mathematics and Computer Science, Hunan Normal University, Changsha, Hunan, China
[3]Department of Electrical and Computer Engineering, North Dakota State University, Fargo, ND, USA
[4]Department of Computer Science, State University of New York, New Paltz, NY, USA

Correspondence
Zheng Xiao, College of Information Science and Engineering, Hunan University, Changsha, Hunan 410082, China.
Email: zxiao@hnu.edu.cn

Summary

Joint service involving several clouds is an emerging form of cloud computing. In hybrid clouds, the schedulers within 1 cloud must not only self-adapt to the job arrival processes and the workload but also mutually adapt to the scheduling polices of other schedulers. However, as a combinatorial optimization problem, scheduling is challenged by the adaptation to those dynamics and uncertain behaviors of the peers. This article studies the collaboration among benevolent clouds that are cooperative in nature and willing to accept jobs from other clouds. We take advantage of machine learning and propose a distributed scheduling mechanism to learn the knowledge of job model, resource performance, and others' policies. Without explicit modeling and prediction, machine learning guides scheduling decisions based on experiences. To examine the performance of our approach, we conducted simulation using the SP2 job workload log of the San Diego Supercomputer Center under a test bed based on agent-based systems—SWARM. The results validate that our approach has much shorter mean response time than 5 typical dynamic scheduling algorithms—opportunistic load balancing, minimum execution time, minimum completion time, switching algorithm, and *k*-percent best. A better collaboration in hybrid cloud is achieved by full adaptation.

KEYWORDS

collaboration, distributed computing, hybrid cloud, machine learning, Q-learning, task scheduling

1 | INTRODUCTION

Cloud computing has emerged as the next-generation platform for hosting business and scientific applications. It is used to virtualize all of the online resources as an on-demand computing model. And it achieves reasonable commercial success. Companies such as Amazon, Google, and Microsoft have deployed a variety of clouds.

Recently, the concept of hybrid cloud is emerging. Cloud collaboration or joint service is of significance to users as well as service providers. First, this could greatly reduce access time and latency especially when a certain cloud is overloaded. Second, another benefit of a hybrid cloud model is the ability to have on-premises computational infrastructure that can support the average workload for your business, while retaining the ability to leverage the other cloud for failover circumstances in which the workload exceeds the computational power of its own cloud component. Third, it makes composition of services possible. Those advantages seem attractive to the cloud owners. They no longer need to build a larger cloud for occasional load peaks, and meanwhile, they can increase users' quality of experience. Many companies pay much attention to hybrid cloud.[1] Vmware has already supported the data or job migration between clouds.[2]

In the applications of clouds,[3] users usually do not care about where and how their jobs get executed. Services are provided by scheduling jobs onto processing elements (PEs) or virtual machines. Scheduling is key to joint service or collaboration among clouds. By scheduling, jobs migrate from 1 platform to another, aiming at improving the system performance.

However, hybrid cloud may involve several autonomous domains. Schedulers in one cloud must not only self-adapt to the job arrival processes and the workload but also mutually or collectively adapt to the scheduling polices of other domains. In this article, the former is named as self-adaptation, while the latter is called mutual adaptation.

In clouds, 3 kinds of uncertainties challenge self-adaptation. First, it is unknown when and how many jobs will arrive. We do not know when and what kind of jobs a user will initiate. Second, it is difficult to know how long a PE will take to execute a job owing to the system dynamics. The performance of a PE may vary in time. Moreover, a job may need to access other networked resources, and the incurred delay is hard to

evaluate. Third, it is hard to determine how long a job must wait before execution. As the system is not dedicated, there may be jobs from other users.

Mutual adaptation brings more challenges to scheduling. Because different schedulers have different job patterns, execution dynamics, and scheduling policies, an efficient collaboration needs to adapt to them. More details can refer to Section 3. This paper focuses on benevolent schedulers, which are cooperative to assist other schedulers, ignoring its cost. Collaboration of benevolent clouds pursues maximization of system performance or full exploitation of the system resources unconditionally. As an online open system, mean response time (MRT) is the metric used for performance evaluation of hybrid cloud. Response time of a job is defined as the sum of the waiting time and execution time.

Most of the existing works barely considered the interaction of schedulers from different clouds. Such works (eg, 2 studies[4,5]) make allocation decisions based on the prediction of the completion time on the PEs in its domain. In addition, heuristics such as genetic algorithms [6] and ant colony optimization [7] could be used to search for the best solution, but such techniques are offline and must have a global view of the whole system.

Hybrid cloud is analogous to a social network, in which users cooperate to complete a job.[8] Users' behaviors are important to the quality of collaboration. García-Cuesta and Iglesias used statistical analysis and subspace learning for user modeling to characterize the computer use behavior.[9] In this work, we take advantage of machine learning and propose a distributed scheduling mechanism to learn the unknown behaviors of the other schedulers. Because of the uncertain job model and the system dynamics, learning happens under a nonstationary environment.[10] So we adopt the methodology of reinforcement learning—a model-free online learning, as indicated in 1 study.[10] Without explicit modeling and prediction, reinforcement learning guides the scheduling decisions based on experiences. Moreover, a layered scheduling model is provided to organize all of the schedulers. The proposed approach learns the knowledge of the job model, resource performance, and schedulers' policies, addressing self-adaptation and mutual adaptation for hybrid clouds. To examine the performance, we conducted simulation studies using the track SP2 job workload log of San Diego Supercomputer Center (SDSC) under a test bed based on agent-based systems—SWARM. The results validate that our approach has much shorter MRT than 5 typical dynamic scheduling algorithms, namely, *opportunistic load balancing* (OLB), *minimum execution time* (MET), *minimum completion time* (MCT), *switching algorithm* (SA), and *k-percent best* (kPB).

The remainder of this article is organized as follows. In Section 2, a layered scheduling model is described. The interaction among the schedulers is analyzed in Section 3. Section 4 provides a learning-based self-adaptive and mutually adaptive scheduling algorithm. A comparison study of our proposed algorithm is presented in Section 5. Section 6 discusses the related work on scheduling, especially on the dynamic scheduling. Section 7 concludes this article.

2 | PROBLEM DESCRIPTION

We use Figure 1 to describe the hybrid clouds. All of the resources from the enterprises or research institutes are connected together using the Internet. However, the resources may belong to different clouds. A PE could be a virtual machine or a physical machine. The set of PEs can be defined as $PE = \{pe_1, pe_2, \dots, pe_n\}$, where n is the number of PEs. Every cloud has one or more schedulers to deal with all the arriving jobs. All schedulers form a set $S = \{s_1, s_2, \dots, s_{|S|}\}$, where $|S|$ is the number of schedulers.

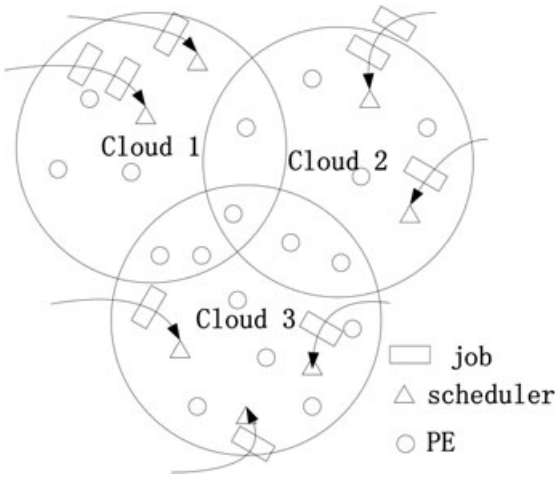

FIGURE 1 An overview of hybrid clouds. PE indicates processing element

Jobs in Figure 1 are independent and initiated by users at random. Assume that hybrid cloud can handle m types of jobs. A type of jobs means that jobs have similar operations but perhaps of a different scale. The set $T = \{t_1, t_2, \dots, t_m\}$ represents all the possible job types. A job of type t_i on scheduler s_j is denoted by $\tau_i^j(k)$, if it arrives at time k. The sequence of jobs arriving at s_j is called a job flow TF_j.

In hybrid clouds, owing to the resources' capacity, the number of jobs per scheduler or PE can accept is limited. When the amount exceeds, this will severely affect the performance of the system and increase the response time of jobs. To solve this problem, jobs may be transferred to other clouds.

A job $\tau_i^j(k)$ can be allocated to the adjacent schedulers or the affiliated PEs. To use resources in hybrid clouds efficiently, shorter response time is expected to improve quality of experience.

Job migration in hybrid clouds can be described by a general framework, a layered scheduling model (see Figure 2). These schedulers that

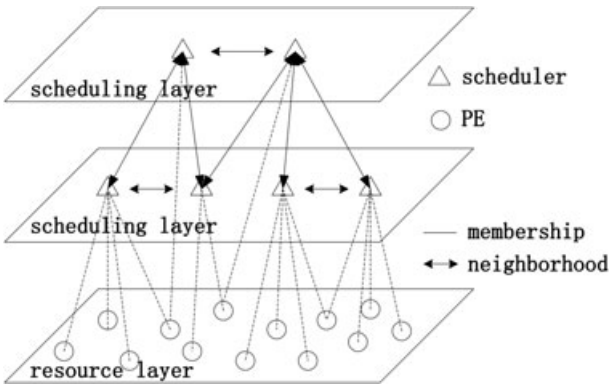

FIGURE 2 A layered scheduling model of hybrid clouds. PE indicates processing element

have more neighbors or cross-domain links are put at higher layers. Through higher layers, a job can span much more schedulers and match a better resource. In Figure 2, all the PEs are located at the resource layer. A PE belongs to at least one or more clouds. Jobs are finally assigned to the PEs for processing. If the PEs can directly be mapped to jobs by a scheduler, the PEs are connected to the scheduler by the dashed lines. A scheduler is able to communicate with the affiliated PEs, so that the scheduler knows the load of the PEs. The schedulers that are connected by lines with arrows are the neighboring schedulers, which can communicate with each other. During scheduling, a job can be assigned to the affiliated PEs or to the neighboring schedulers. If no assignment is satisfactory, then the scheduler will hand over the job to the neighbor at a higher layer. The scheme proposed in Section 4 can be applied to all the scheduling layers.

3 | MOTIVATION

There are 2 kinds of influence that might degrade the efficiency of collaboration, given in Figure 3.

1. *Master-slave influence*—As shown in Figure 3A, TF_1 and TF_2 are job flows on schedulers s_1 and s_2, respectively. We call TF_1 as the master flow, and TF_2 as the slave flow. s_1 immigrates a certain kind of tasks to the neighbor s_2 for collaboration. The MRT caused by this collaboration depends not only on the load of s_2 but also on the jobs arriving at s_2 in the future. For example, if at time k, s_2 has a lighter load, but in TF_2 a large batch of jobs need to be processed by s_2, the MRT increases because s_1 delivered jobs to s_2.
2. *Non–master-slave influence*—Figure 3B illustrates the other influence. Suppose that at time k, the scheduler s_3 executes jobs of type t_i faster than s_4. Usually, s_1 tends to assign jobs of type t_i to s_3, obtaining faster response. However, the allocation neglects influence from the job flow TF_2. If after time k, s_2 has a large number of jobs for s_3 too, perhaps it will be much more reasonable that s_1 assigns jobs to the suboptimal scheduler s_4. Although the response time for s_1 may be prolonged, jobs on s_2 will receive quicker response, resulting in shorter MRT of the system.

In brief, efficient collaboration means job allocation should adapt to the job arrival patterns, execution dynamics, and scheduling behavior of other schedulers.

4 | LEARNING AND ADAPTIVE SCHEDULING

The background of Markov decision process (MDP) and Q-learning is given before describing why and how our algorithm is proposed.

4.1 | Background of MDP and Q-learning

Figure 4 shows a 3 × 4 grid world, in which there are 12 states. $ST = \{st_i, i = 1,2,\dots,12\}$ is the set of states. At each state, 4 actions could be selected. $AC = \{UP, DOWN, LEFT, RIGHT\}$ is the set of optional actions. At some states like the border, some actions are infeasible. A function $R(st_i \in ST, ac_j \in AC) \in \Re$ represents the reward obtained by choosing action ac_j at state st_i. The problem is to find a path from st_1 to st_{12} by taking a sequence of actions such that the accumulative reward is maximal. At each state, a decision should be made on which action to take. The solution is a sequence of decisions. Such a problem is called a *sequential decision process* in operational research.

In the sequential decision process, the state st^{t+1} at the $(t + 1)$th moment depends on the previous states and the action ac^t taken at the tth moment. The transition between states is defined by $st^{t+1} = \Gamma(st^1, \dots, st^t, ac^t)$, where $st^t \in ST$, $ac^t \in AC$. If st^{t+1} only depends on the current state st^t and action ac^t, ie, Markov property, the sequential decision process becomes MDP.[11] In MDP, the state transition function becomes $st^{t+1} = \Gamma(st^t, ac^t)$.

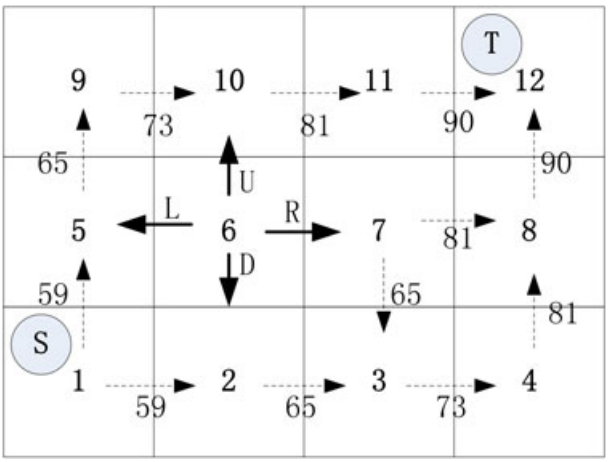

FIGURE 4 A 3 × 4 grid world

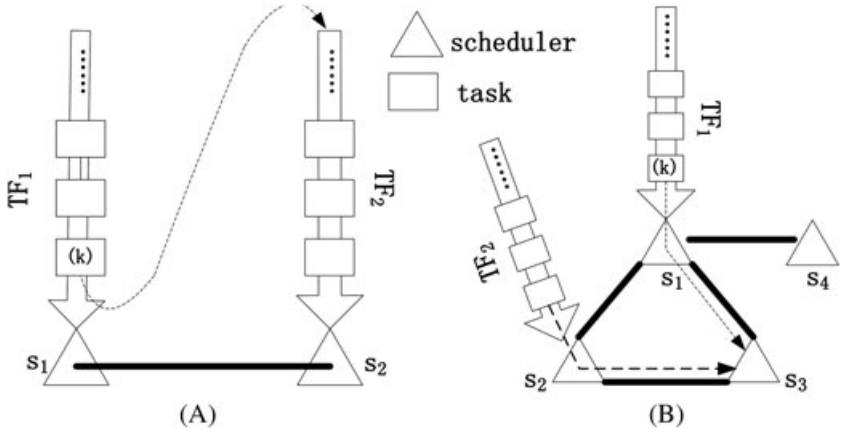

FIGURE 3 Classification of the mutual influence. A, master-slave influence. B, Non–master-slave influence

The grid world problem is a MDP. Suppose there is a path that reaches the terminal state st_{12} at the moment τ. Then the accumulative reward AR is as follows.

$$\begin{aligned} AR &= r^1 + r^2 + \cdots + r^\tau = \sum_{t=1}^{\tau} R\left(st^t, ac^t\right) \\ &= \sum_{t=1}^{\tau} R\left(\Gamma\left(st^{t-1}, ac^{t-1}\right), ac^t\right), \end{aligned} \tag{1}$$

where $st^1 = st_1$ and $st^\tau = st_{12}$.

To find the maximal AR, we defined 2 functions for iterative solution: $Q(st_i, ac_j)$, which means the accumulative reward by taking action ac_j at state st_i, and $V(st_i)$, which means the maximal accumulative reward at st_i.

$$Q\left(st_i, ac_j\right) = R\left(st_i, ac_j\right) + V\left(\Gamma\left(t_i, ac_j\right)\right) \tag{2}$$

$$V(st_i) = \max_j Q\left(t_i, ac_j\right) \tag{3}$$

The problem becomes to compute $V(st_1)$. Equations 2 and 3 give a naive Q-learning algorithm. Assume for $\forall i$ and $\forall j$, $R(st_i, ac_j) = 0$. Initially, let $Q(st_i, ac_j) = 0$ and $V(st_i) = 0$ except $V(st_{12}) = 100$. From the initial state st_1, take actions at random and then update Q and V. If the grid world is explored fully, $V(st_{12}) = 100$ will propagate backward to st_1. In this example, all Q and V equal to 100. This result shows that no matter what decision is made at each state, the terminal state will be reached on enough time. However, it is often desirable that the terminal state is reached as soon as possible or the maximal accumulative reward is obtained in shorter time. An improved Q-learning[12] is proposed. The less importance is assigned to the future decisions while the weight of immediate reward gets more. So Q is discounted by a discounted factor $\lambda \in [0,1]$.

$$Q\left(t_i, ac_j\right) = R\left(t_i, ac_j\right) + \lambda \cdot V\left(\Gamma\left(t_i, ac_j\right)\right) \tag{4}$$

Based on Equations 3 and 4, the discounted accumulative reward is defined.

$$\begin{aligned} DAR &= r^1 + \lambda r^2 + \cdots + \lambda^{\tau-1} r^\tau = \sum_{t=1}^{\tau} \lambda^{t-1} R\left(st^t, ac^t\right) \\ &= \sum_{t=1}^{\tau} \lambda^{t-1} R\left(\Gamma\left(st^{t-1}, ac^{t-1}\right), ac^t\right) \end{aligned} \tag{5}$$

The numbers in Figure 4 are the corresponding Q values according to Equations 4 and 3 if $\lambda = 0.9$. When the action $\arg\max_{ac_j} Q\left(t_i, ac_j\right)$ is taken, the shortest path is found.

It is not always the case that the unique state is transferred to by ac_j at st_i. Because the environment is full of uncertainty, the transition function Γ is a subset of ST, ie, $\Gamma \subseteq ST$. $PD(\Gamma)$ denotes the probabilistic distribution over the potentially transferred states. The iterations become the expected discounted accumulative reward as Equation 6 shows.

$$\begin{aligned} \bar{Q}\left(st_i, ac_j\right) &= R\left(st_i, ac_j\right) + \lambda \cdot E\left(V\left(\Gamma\left(st_i, ac_j\right)\right)\right) \\ &= R\left(st_i, ac_j\right) + \lambda \cdot \sum_{st_k \in \Gamma\left(st_i, ac_j\right)} \mathrm{PD}\left(st_k | st_i, ac_j\right) \cdot \bar{V}(st_k) \\ \bar{V}(st_i) &= \max_{ac_j} \bar{Q}\left(t_i, ac_j\right) \end{aligned} \tag{6}$$

Under the iterations of Equation 6, the objective becomes the expected discounted accumulative reward $E(DAR)$.

$$E(DAR) = \sum_t \lambda^{t-1} \sum_{st^t \in \Gamma\left(st^{t-1}, ac^{t-1}\right)} PD\left(st^t | st^{t-1}, ac^{t-1}\right) \cdot R\left(st^t, ac^t\right) \tag{7}$$

If the functions Γ and $PD(\Gamma)$ are a priori knowledge, the traditional Q-learning method can be used to find the maximal $E(DAR)$. Unfortunately, environment is not easy to model. A practical online Q-learning,[11] 1 method of model-free machine learning, is proposed to solve the MDP problem. It defines the iterations as Equation 8.

$$\begin{aligned} \hat{Q}\left(st_i, ac_j\right) &= R\left(st_i, ac_j\right) + \lambda \hat{V}(st')), st' \in \Gamma\left(st_i, ac_j\right) \\ \hat{V}(st_i) &= \max_j \hat{Q}\left(st_i, ac_j\right) \end{aligned} \tag{8}$$

That theory establishes that $\hat{Q}$ and $\hat{V}$ converge to $\bar{Q}$ and $\bar{V}$ on condition of fully traversal of the entire "world." Fully traversal means that all the states and actions are explored and visited sufficiently. So this theory is also called an error and attempt approach. The ε-greedy exploration is a widely used policy for that. The exploration probability p_e is preset. The action is selected randomly by lower probability than p_e; otherwise, the action $\arg\max_{ac_j} \hat{Q}\left(t_i, ac_j\right)$ is selected.

Equation 8 is the form of Q-learning we used. It is model free. The learning process is able to adapt to the transition function and its probability.

4.2 | Problem modeling by MDP

In distributed scheduling, jobs arrive at random. Aiming to minimize the MRT, the allocation of the current jobs not only depends on the allocation of the jobs already arrived but is also influenced by the jobs to arrive later. If just assigning jobs to the PEs that have the shortest response time and neglecting the scheduling of the later arriving jobs, the MRT may not be optimal, as discussed earlier. Therefore, if a single job's allocation is regarded as a decision step, scheduling is made up of a sequence of related steps.

Furthermore, the state transition only depends on what jobs are being scheduled and the workload on the PEs. The scheduling problem is a MDP, which is defined by a quadruple (ST, AC, Γ, R). In the context of our scheduling problem, the particular MDP model is defined on each scheduler as follows.

- $ST = \{st_1, st_2, \ldots, st_{|ST|}\}$. If a job is being allocated by the scheduler s_j, the state includes the job type and the workload on all the possible units. We use PU to denote the set of possible units to which the job can be assigned. The elements in PU can be divided into 4 subsets, ie, the set of affiliated PEs (APE), the set of s_j's neighboring schedulers at the lower layer (LS), the set of s_j's neighboring schedulers at the same layer (NS), and the set of s_j's neighboring schedulers at the higher layer (HS). PU = APE ∪ LS ∪ NS ∪ HS, and 1 state in ST is described by the vector below:

  $$\left(TYPE, LOAD_{(1\cdots|APE|)}, LOAD_{(1\cdots|LS|)}, LOAD_{(1\cdots|NS|)}, LOAD_{(1\cdots|HS|)}\right).$$

  The load of a processing unit in PU is the ready time for the current job. However, the execution time of a job already waiting for execution is hard to estimate, so the number of jobs already waiting for execution is used as the load metric.

  a. For a PE, the load is defined as the number of jobs already waiting for execution.

b. For a scheduler at the first layer, the load is defined as the maximal load of the affiliated PEs, which is a pessimistic strategy.
c. For a scheduler at the higher layers, the load is defined as the minimal load of the schedulers at the lower layers.

In a large-scale system, the number of states could be numerous, which leads to the curse of dimensio nality. For this problem, the approach of state aggregation[13] is used. The parameter β of aggregation granularity is used for clustering. The continuous values of the size β are clustered into 1 state. For example, the interval [0, 100) includes 100 states if $\beta = 1$ and 20 if $\beta = 5$. The state space is reduced through aggregation by increasing β.

- $AC = \{(st_i, PU_j) | st_i \in ST, PU_j \in PU\}$ is the set of all possible allocations. The element $AC_{ij} = (st_i, PU_j)$ in AC is a pair of states and possible units. It means that under the state st_i, a job is assigned to the PU_j.
- $\Gamma : ST \times B \rightarrow ST$ is a transition function. It describes the relation among the states. The types of jobs are nondeterministic and follow a certain discrete probability distribution. Therefore, the *TYPE* in the next state is a stochastic variable. Moreover, the *LOAD* on *PU* is uncertain because the policies of other schedulers are unknown. Therefore, the transited state is not a deterministic state. The transition function uses a probability distribution $PD(ST)$ over the potentially transferred states. $PD(ST) = \{P(st' \mid st_i, b_{ij}) \mid \sum_{st' \in ST} P(st' \mid st_i, b_{ij}) = 1, 0 \leqslant P(st' \mid st_i, b_{ij}) \leqslant 1\}$.
- $R : \mathrm{ST} \times B \rightarrow \mathfrak{R}$ is the immediate reward function and $r \in \mathfrak{R}$ represents the immediate reward when taking the allocation $ac_{ij} \in AC$ under the state $st_i \in ST$. The shorter the response time a single job has, the larger the reward r it returns. Therefore, an inverse proportional function could be used to compute the value of r.

In the above MDP model, the job arrival process and the mutual influence of the schedulers are implied in the state transition function Γ. The execution process of the individual jobs is implied by the immediate reward function R.

We aim to minimize the MRT determined by all of the allocations of the job flows. The current allocation should consider its own response time and the influence of the forthcoming allocations. Such an influence becomes weaker on the subsequently arriving jobs. Therefore, the discounted accumulative reward is used as the objective function. Starting from the tth allocation, it can be defined below:

$$DAR(t) = r^t + \lambda r^{t+1} + \lambda^2 r^{t+2} + \cdots = \sum_{i=0}^{\infty} \lambda^i r^{t+i}, \ t \geqslant 1, \tag{9}$$

where $\lambda \in [0,1]$. The bigger the value is, the more importance the decision puts on the subsequent allocations. It could be set empirically.

Equation 9 relies on the state transition function Γ and the immediate reward function R, which actually relate to job arrival, execution process, and policies of other schedulers. As Section 4.1 described, Q-learning approach can be used to maximize *DAR*. It learns the historical online allocations to gradually adapt to uncertainties from environment and other peers without any prior knowledge.

4.3 | A Q-learning approach

To maximize *DAR*, it is necessary to have complete information about the arrived jobs and the jobs to arrive, ie, the knowledge of the arrival and execution processes, as well as the policies of other schedulers. Q-learning does not need to model job arrival and execution processes and learns from the historical allocations as the training samples.

Because the state transition function Γ is a probabilistic event and not a priori knowledge, we can define the iterations for Q-learning according to Equation 6.

$$\begin{aligned} \bar{Q}(t_i, ac_{ij}) &= R(st_i, ac_{ij}) + \lambda \mathrm{E}\left(V\left(\Gamma\left(st_i, ac_{ij}\right)\right)\right] \\ &= R(st_i, b_{ij}) + \lambda \sum_{st' \in \Gamma(st_i, ac_{ij})} \mathrm{PD}(st' | st_i, ac_{ij}) V(st') \\ \bar{V}(st_i) &= \max_j Q\left(st_i, ac_{ij}\right) \end{aligned} \tag{10}$$

The optimal scheduling scheme π is the allocation that has the maximal $\bar{Q}$:

$$\pi(st_i) = \arg\max_j Q\left(st_i, ac_{ij}\right). \tag{11}$$

Although to solve it, we need to know $P(st' | st_i, ac_{ij})$ and $\Gamma(st_i, ac_{ij})$, which are related to job arrival, others' scheduling policies, and execution process. But the knowledge of them is hard to obtain. To build their models is 1 solution to get such knowledge, but it is a nontrivial job.

Q-learning of [11] is a promising approach. During learning, ε-greedy exploration is used to visit the states and actions sufficiently. After each decision is made and completed, the transferred state and reward are used to train $\hat{Q}$ and $\hat{V}$. It is pointed out in Section 4.1 that Q-learning works on premise of sufficient exploration. There is no evidence that ε-greedy exploration is totally competent. So another mechanism, dynamic learning rate, is introduced to guarantee sufficient exploration. Equation 12 gives learning rule of the lth iteration:

$$\begin{aligned} \hat{Q}_l\left(st_i, ac_{ij}\right) &\leftarrow (1 - \alpha_l)\hat{Q}_{l-1}\left(st_i, ac_{ij}\right) + \alpha_l(R\left(st_i, ac_{ij}\right) + \lambda \hat{V}_{l-1}(st')), \\ \hat{V}_l(st_i) &\leftarrow \max_j \hat{Q}_l\left(st_i, ac_{ij}\right). \end{aligned} \tag{12}$$

In Equation 12, $st' \in \Gamma(st_i, ac_{ij})$ is the transferred state now. The immediate reward $R(st_i, ac_{ij})$ is the reciprocal of the interval from arrival of a job until its completion at PU_j. $\alpha_l (0 \leqslant \alpha_l \leqslant 1)$ is the learning rate for the lth iteration, which is defined by Equation 13:

$$\alpha_l = \frac{1}{1 + visits_l\left(st_i, ac_{ij}\right)}, \tag{13}$$

where $visits_l(st_i, ac_{ij})$ is the number of visits of the pair (st_i, b_{ij}) during the l iterations. If a state or action is visited frequently, its learning rate gets lower. In other words, the less frequently visited states or actions get more chance to learn. This mechanism helps sufficient and fair exploration.

Next, we discuss how our approach takes effect. In distributed scheduling, there are a number of schedulers and task flows. A scheduler can allocate a job to the affiliated PEs or the adjacent schedulers. The complete algorithm is given in Algorithm 1. The learning rule given by Equation 12 is extended to adapt to the task flows and policies of other schedulers.

This algorithm is an online learning algorithm. It learns the allocation scheme after a job is allocated and executed. It stops when the changes of Q-function are lower than a customized threshold. It does not need to model job arrival, evaluate execution cost, and predict others' policies. However, there is a weakness; ie, jobs used for training before its convergence may be allocated to inappropriate processing units, and users have to pay for the worse performance.

Algorithm 1 Reinforcement Learning based Algorithm

(1) Initialization
  for each $st_i \in ST$ and each $b_{ij} \in B$ **do**
    $Q(st_i, b_{ij}) = 0$;
    $visits(st_i, b_{ij}) = 0$;
  end for
(2) Action selection for task t_i
  if explore with probability p_e **then**
    select allocation b_{ij} randomly by uniform distribution;
  else
    select allocation b_{ij} by the optimal scheme $\pi(st_i) = argmax_j Q(st_i, b_{ij})$;
  end if
(3) Learning
  $\alpha = 1/(1 + visits(st_i, b_{ij}))$;
  update Q: $Q(st_i, b_{ij}) = (1 - \alpha)Q(st_i, b_{ij}) + \alpha(R(st_i, b_{ij}) + \gamma V(\Gamma(st_i, b_{ij})))$;
  update V: $V(st_i) = \max_j Q(st_i, b_{ij})$;
(4) Updating
  $visits(st_i, b_{ij}) = visits(st_i, b_{ij}) + 1$;
  if the convergence criterion is satisfied **then**
    return π;
  else
    goto (2);
  end if

Our algorithm works seamlessly in hierarchic scenario. In the definition of states, we treat the PEs or the adjacent schedulers as processing units, although the load is counted differently. When a job is ready for allocation, it compares $\hat{Q}$ of all the adjacent entities, including any in *APE*, *LS*, *NS*, and *HS*. If a entity A in *HS* has the maximal $\hat{Q}$, then the job is allocated to it. Maximum means that cloud has light load or high capability, or it knows other clouds have better performance. In the same way, A may pass the job to B in A's *LS*, until it finally reaches a PE. In fact, the hierarchy is flatted.

During deployment, the key problem is how to get the immediate reward when a job has been allocated to its neighboring schedulers. If a scheduler only interacts with its affiliated PEs, the PE returns the response time when a job is completed. But in our case, a job can be finally allocated to a PE through several schedulers. This forms a scheduling chain as Figure 5 shows. The cost brought by forwarding a job to a processing unit is stochastic at each stage of scheduling. The network delay is not ignorable and contributes to the response time. Besides, there is a synchronizing problem among the schedulers, especially locating in different domains. We use a timestamp mechanism to solve the asynchronous problem. Each scheduler gives a timestamp when a job is scheduled. When a job is completed, a notification with a timestamp is returned to the schedulers. Then all the schedulers on the scheduling chain compute the immediate rewards and trigger their learning processes.

In Figure 5, a job is submitted by a user and scheduled 3 times before execution on a PE. Steps 2, 4, and 6 indicate that at each scheduling, a timestamp is appended when the job is forwarded. The schedulers take charge of recording its own scheduling time and collecting the post-scheduling time. If the job is completed, a timestamped notification is generated in step 8 and returned to its previous schedulers in order of 9, 10, and 11. Upon receiving it, the schedulers compute the immediate reward and launch learning.

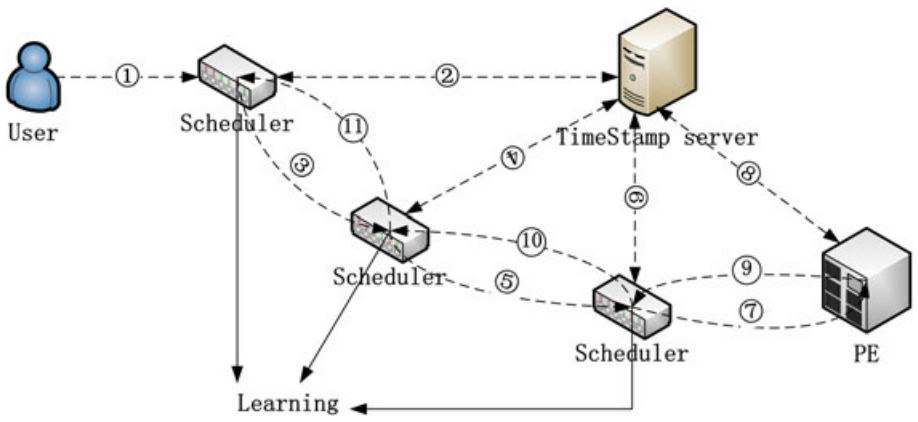

FIGURE 5 Synchronizing in distributed scheduling. PE indicates processing element

5 | SIMULATION USING SWARM

In this section, we will validate our proposed algorithm under the track SP2 job workload log of SDSC.[14] Before comparing its performance, some involved parameters are analyzed first, and an empirical setting is used in the following experiments of performance comparison. Several typical algorithms are selected as the benchmark. The MRT and makespan are used as the performance metrics.

A test bed of our lab developed based on SWARM is used to simulate a distributed environment. SWARM is a complex, adaptive, system-oriented, agent-based modeling tool initiated at Santa Fe Institute. This tool easily builds a multiagent system with multiple roles like schedulers and PEs. This test bed only helps visualize the simulation.

5.1 | Data set and configuration

Two data sets are used to simulate the system structure as well as the possible dynamics of jobs and resources.

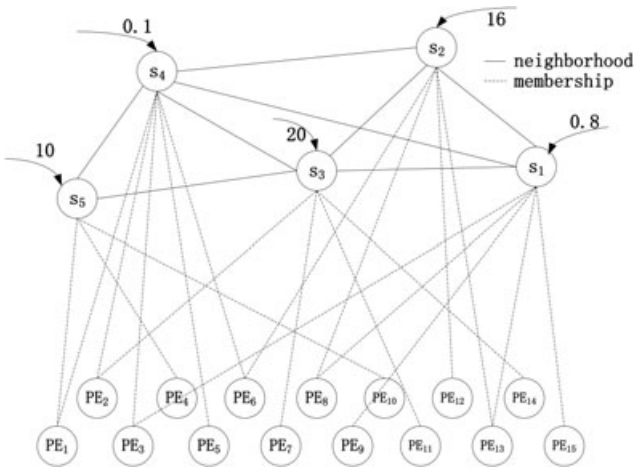

FIGURE 6 A small-scale distributed system

(A) Queueing theory–based data set

We use a synthetic and small-scale distributed system as Figure 6 shows. For simplicity, it is composed of 5 schedulers in the scheduling layer and 15 PEs in the resource layer. The job flow on each scheduler follows a Poisson arrival process with different arrival rates. Assume 10 types of jobs might arrive. The probability of each type is 0.1 equally. Their execution time on each PE by unit time is a random integer within the range 5 to 30, given in Table 1.

(B) SP2 log–based data set

The real-world workload, SP2 log of SDSC,[14] is used to further validate our algorithm. SP2 cluster corresponds to a microhybrid cloud in which the job migration cost between clouds is assumed to be equal. There are 128 nodes or PEs. The queues to accommodate the arriving jobs are 7. Each queue corresponds to a scheduler. However, the IBM SP2 is not a real distributed environment, since the schedulers have a global view of all the PEs. To simulate the collaboration of the schedulers, the 128 PEs belong to the 7 schedulers randomly. Suppose the 7 schedulers are full connected peers at the single scheduling layer. As a result, a scheduler sometimes has to pass a job to other schedulers, because the fast resource may not belong to itself. The jobs in the log are assumed to be atomic. According to the log, the arriving times of all jobs are provided. So a real job arrival process is simulated in our experiments. There are 73496 records in total, which are divided into 3-folds for k-fold cross-validation.

5.2 | Parameter analysis

The queueing theory–based data set is used to analyze and estimate those 2 parameters. There are 2 parameters in our algorithm, ie, state aggregation granularity β and discount factor λ. Their definitions can be referred to in Section 4. The optimal settings of β and λ vary as the different environments, depending on the network architecture, the workload pattern, the number of PEs, etc. Although β and λ are not optimal for all cases, this is one feasible way of parameter evaluation.

TABLE 1 Execution time of 10 task types on all PEs (unit time)

Type	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}
PE_1	25	27	8	27	20	7	11	18	29	17
PE_2	8	15	27	24	28	21	5	26	21	21
PE_3	22	5	11	6	7	25	22	12	5	24
PE_4	17	16	21	22	23	11	21	21	7	19
PE_5	23	11	17	22	27	28	18	8	11	11
PE_6	13	9	11	20	16	13	25	19	27	14
PE_7	6	6	18	24	28	8	19	16	13	18
PE_8	20	11	21	22	23	16	7	10	8	6
PE_9	7	29	5	24	25	26	7	14	25	11
PE_{10}	8	26	19	18	8	26	20	13	15	9
PE_{11}	15	6	27	28	17	17	13	27	7	15
PE_{12}	8	28	28	19	6	10	13	25	6	21
PE_{13}	18	12	23	9	22	9	14	20	7	15
PE_{14}	12	17	17	25	24	21	14	25	13	20
PE_{15}	10	12	16	10	26	9	10	9	15	9

Abbreviation: PE, processing element.

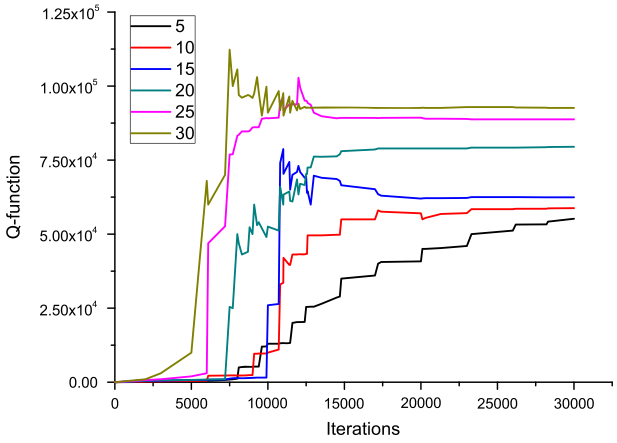

FIGURE 7 Convergence with different state aggregation granularity

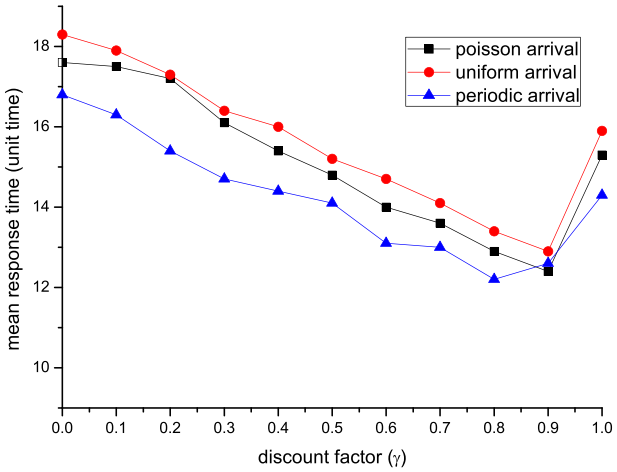

FIGURE 8 Mean response time with different discount factors

- *State aggregation granularity*—We use state aggregation to avoid the curse of dimensionality. A number of neighboring states are categorized as a single state. If the aggregation size is small, the number of states will be large and lead to a slow convergence. But if it is too big, a state cannot approximate the real system load.
- *Discount factor*—It determines how much importance the current allocation lays on the subsequent allocations. The bigger it is, the more weight is there on the future allocations. Instead, if it approaches 0, the allocation only considers the current load and becomes greedy scheduling.

From the statements above, β and λ play a role in 2 indispensable attributes: convergence and MRT.

In the first group of experiments, convergence is analyzed when the discounted factor $\lambda = 0.9$ and the state aggregation granularity β is set by several different values. The functions Q and V ought to converge. Figure 7 depicts the curve of a certain Q-function at each iteration. The number of states has some influence on the convergence speed. It is observed that as β decreases, it converges more slowly, because more states have to be visited and learned. With $\beta = 5$, it even does not converge within 30000 iterations. But if the granularity is too big, many neighboring states are clustered into a single state. The approximation accuracy degrades. To trade off the convergence speed and the state space size, the aggregation granularity β is set to 20 for the experiments of performance comparison.

In the second group of experiments, we attempt to analyze the influence of discount factor λ. This parameter reflects the relationship between current and successive allocations. Figure 8 shows the MRT with the discount factor from 0 to 1. More weight is attached to the future allocations if λ gets higher. The response time includes the waiting and execution time. And it is averaged over 1000 jobs as the MRT. The environment keeps the same as the first group of experiments. Except for the Poisson arrival process, the uniform and periodic arrival processes are also tested. In these arrival processes, the arrival rate is unchanged as in Figure 6.

In Figure 8, the best performance happens when λ is between 0.8 and 1. So by a rough estimation, we use 0.9 as the discount factor in the following experiments.

5.3 | Performance analysis

Since our method schedules a job to the appropriate resource immediately after its submission, the proposed method falls into immediate mode scheduling scheme.[15] Therefore, 5 immediate mode scheduling algorithms, OLB, MET, MCT, SA, and kPB, are selected to be compared with our proposed scheduling method. They are used as benchmarks in many research works.

- OLB: This algorithm assigns each task to the earliest idle resource without any consideration about the execution time of the task on the resource. If two or more resources are idle, then a resource is selected arbitrarily. The intuition behind OLB is to keep all resources as busy as possible. One advantage of OLB is its simplicity, but because OLB does not take the task execution times into account, the resulting schedule is not optimal.
- MET: This algorithm assigns each task to a resource that results in the least execution time for that task, regardless of that machine's availability. As a task arrives, all the resources in the environment are examined to determine the resource that gives the MET for the task. Therefore, the motivation behind MET is to give each task to its best resource. But allocating task without considering resource availability results in load imbalance.
- MCT: This algorithm assigns a task to the resource, yielding the earliest completion time (ready time of resource + task execution time on that resource) for that task. When a task arrives in the environment, all available resources are examined to determine the resource that yields the smallest completion time for the task. In MCT, a task could be assigned to a resource that does not have the smallest execution time for that task. The intuition behind MCT is to combine the advantages of OLB and MET, while avoiding their drawbacks. This method is also known as fast greedy, originally proposed for SmartNet system.
- SA: The MET method has a potential drawback in that it can lead to load imbalance across resources by assigning many more tasks to some resources than to the others since it blindly looks at execution times of the tasks without considering the ready time of the resources. On the other hand, the MCT heuristic assigns tasks to

resources to achieve earliest completion time, thereby ensuring load balance but does not necessarily minimize the execution times of the tasks. The SA tries to overcome some limitations of MET and MCT methods by combining their best features. Here, the idea is to first use the MCT until a threshold of balance is obtained followed by MET, which creates the load imbalance by assigning tasks on faster resources. More precisely, let r_{max} be the maximum ready time and r_{min} be the minimum ready time; the load-balancing factor is then $r = r_{\min}/r_{\max}$, which takes values in range [0, 1]. It is obvious that for $r = 1$, we have a perfect load balancing, and if $r = 0$, then there exists at least 1 idle resource. Further, 2 threshold values, r_l and r_h, $0 \leqslant r_l \leqslant r_h \leqslant 1$ are used to control the order in which MCT and MET are applied. Initially, r is set to 0 so that SA starts allocating tasks according to MCT until r becomes greater than r_h; after that, MET is activated so that r becomes smaller than r_l, and a new cycle starts again until all tasks are allocated.

- kPB: This method also tries to combine the best features of MCT and MET simultaneously instead of cyclic manner. In this method, only k percentage of best resources, considering their service times, are chosen while assigning the tasks. For a particular task, a resource that gives MCT is selected from the k kPB resources instead of all possible resources. It should be noted that for $k = 100$, kPB behaves as MCT, and for $k =$ (100/total number of resources), it acts as MET.

The r_l and r_h parameters in SA algorithm and k factor in kPB algorithm have been set to 0.1, 0.3, and 0.5, respectively. These values are approximately the best ones in which the results obtained from applying them are very reasonable and acceptable.[16]

In this section, performance of the proposed method is compared with that of the aforementioned 5 typical scheduling algorithms regarding 2 parameters: MRT and total makespan. The response time of a job includes the waiting and execution times. After its arrival, a job has to wait for allocation and execution. The first metric, MRT, is the sum of response time of all jobs divided by the number of jobs. The second comparison metric, total makespan, can be computed as the maximum time among all resources' ready times. Actually, this is the time that the distributed system takes to finish all the jobs. Besides, its extendability is also validated.

5.3.1 | Mean response time

Mean response time is the objective of our algorithm. It is investigated under 3 cases, namely, (1) queueing theory–based data set, (2) SP2 log–based data set with fixed execution time, and (3) SP2 log–based data set with stochastic execution time.

In case (1), the jobs are classified into types by the run-time column in the log. If the run time is close, the jobs are partitioned in 1 type. According to the order of magnitude, 47 types are used in the experiments. Then, for each type, we use a random number generator that follows an exponential distribution to generate the execution time on the 128 PEs. Owing to the constraint of the length, the corresponding table is not provided here.

In case (2), the execution time of a job is deterministic before scheduling. However, in real distributed environment, network delay is dynamic. So the execution time of a job is a stochastic variable.

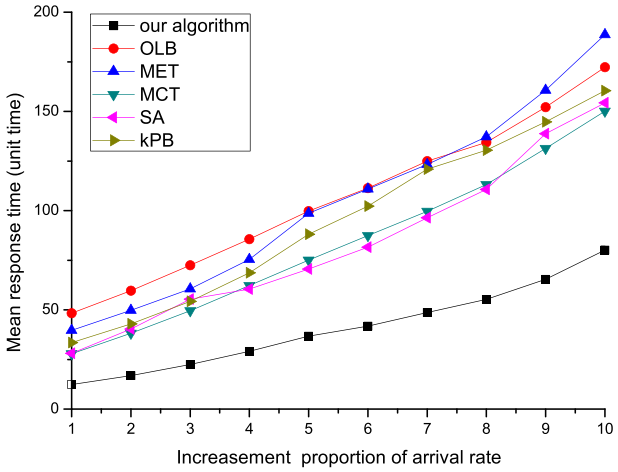

FIGURE 9 Mean response time of queueing theory–based data set. kPB indicates k-percent best; MCT, minimum completion time; MET, minimum execution time; OLB, opportunistic load balancing; SA, switching algorithm

In this group of experiments, it is supposed that the execution time follows an exponential distribution with the service rate μ. A service rate is appointed to all of the PEs for each job type. Algorithms such as MET, MCT, SA, and kPB need the execution time information. Assume that these algorithms have an exact prediction function with the same distribution to estimate the execution time when scheduling.

In the latter 2 cases, the reported MRTs and makespans in Figures 10 and 11 are averaged over 3 experiments in which 1-fold is selected as the test data set and the remaining 2-folds are used as the training data set. Because the execution time in case (3) is stochastic, the corresponding experiments of Figures 10B and 11B repeat 50 times, and the 95% confidence intervals are depicted.

Figures 9 and 10 compare the proposed algorithm with the 5 benchmarks regarding MRT metric under the 3 cases, respectively. Figure 9 shows MRTs in the way that the arrival process keeps Poisson but the arrival rate increases once each time.

The OLB, MET, MCT, SA, and kPB are heuristic algorithms. The OLB and MET only take partial response time of a job into account. Owing to severe load imbalance, their MRTs are highest. Especially, the workload in cases (2) and (3) is heavy; MET always assigns jobs to the best resources, which are blocked. So MET in Figure 10 is the worst.

The heuristic of MCT is minimizing the completion (response) time of current job. But it ignores the response time of subsequent jobs. The shortest response time of a single job does not mean best performance on average. The SA and kPB are tradeoff of MET and MCT, less greedy than MCT and less conservative than MET.

The benchmarks' MRTs are always higher than the MRT of the proposed method in any cases. Unlike them, our algorithm is not shortsighted. From its objective function Equation 9 of a single allocation, our method considers not only the response time of current job but also the influence of the subsequent allocations. Just because each individual allocation is based on the whole task flow, not on a single job, does not mean the MRT of our algorithm is

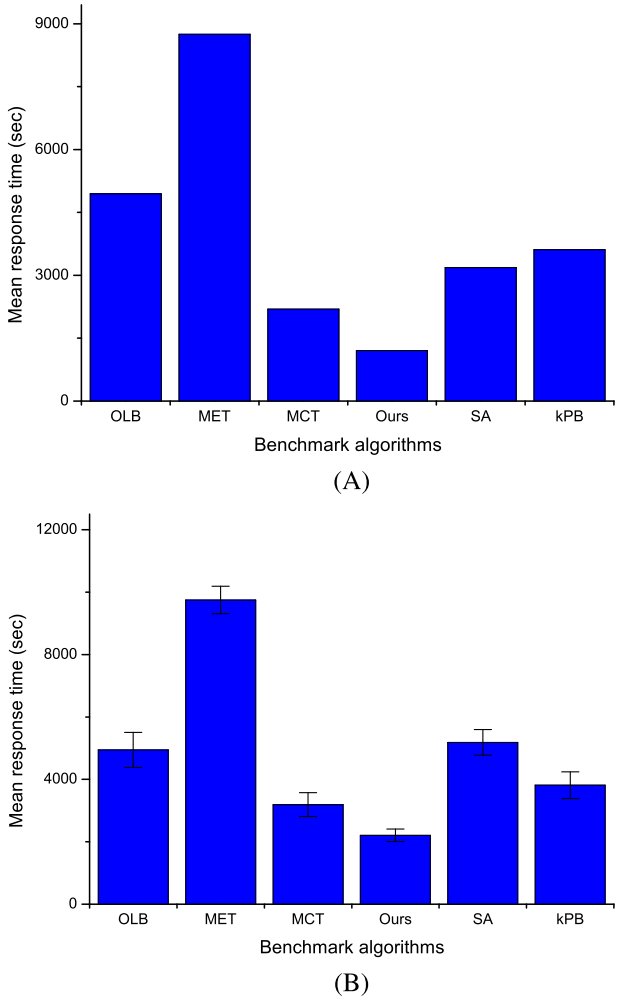

FIGURE 10 Mean response time of SP2 log–based data set. A, Fixed execution time. B, Stochastic execution time. kPB indicates *k*-percent best; MCT, minimum completion time; MET, minimum execution time; OLB, opportunistic load balancing; SA, switching algorithm

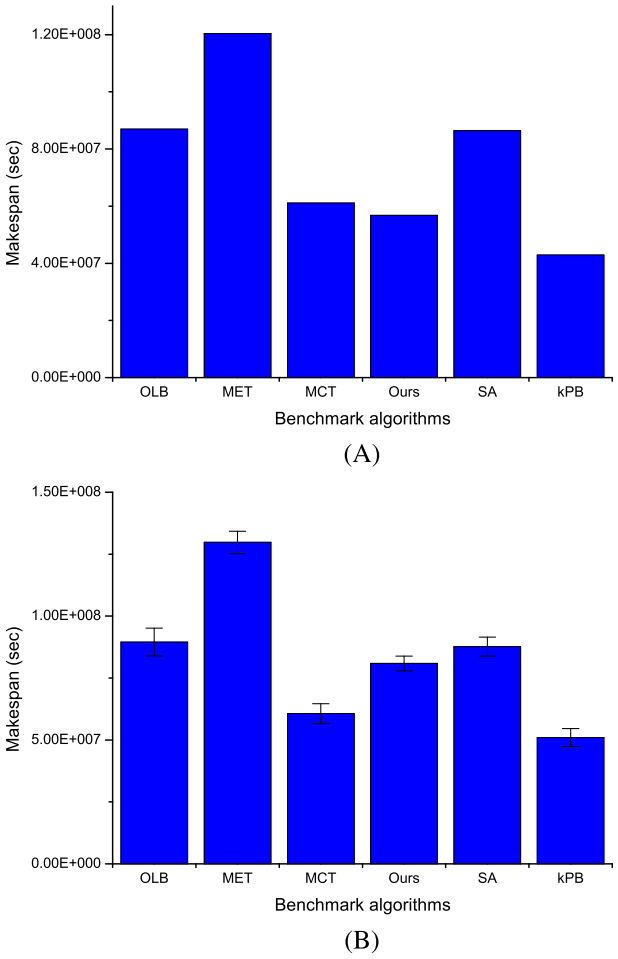

FIGURE 11 Makespan of SP2 log–based data set. A, Fixed execution time. B, Stochastic execution time. kPB indicates *k*-percent best; MCT, minimum completion time; MET, minimum execution time; OLB, opportunistic load balancing; SA, switching algorithm

quite shortened. The interinfluence of individual task flows plays a role in optimizing MRT. Actually, if $\lambda = 0$, our algorithm becomes MCT. As the arrival rate increases and system load gets heavier in case (1), MRT gets higher and higher. More time has to be spent in waiting.

In addition, compared with Figure 10A, MRTs of the benchmarks in Figure 10B increase in a larger magnitude than that of our algorithm. Because of the stochastic execution time, jobs' execution time is not identical as predicted. But our algorithm is not on basis of any prediction model, so its MRT drops less severely. Furthermore, our algorithm has a smaller confidence interval. It verifies that our algorithm is capable of adapting to dynamics.

5.3.2 | Makespan

Figure 11 compares the proposed algorithm with the 5 benchmarks regarding the total makespan metric under cases (1) and (2), respectively. It is not surprising that OLB and MET are worst, because of load imbalance. kPB assigns jobs to the best resources, speeding the execution of jobs. As a result, its makespan is shortest. The reason why makespan of SA is much worse than MCT is not clear, perhaps resulting from the frequent switch between MET and MCT.

Our algorithm does not aim to optimize makespan, so it does not perform its best. However, the first item of Equation 9 reflects the load of resources. If this item is small when a resource is overloaded, it has the trend to decrease the reward, restraining load imbalance. So makespan of our algorithm is equivalent to that of MCT in Figure 11A. When the execution time becomes uncertain in case (3), our algorithm gets worse than MCT as Figure 11B shows. This is because the dynamic execution time of subsequent jobs weakens the restraint of load imbalance in contrast with the fixed execution time. And it does not have an overwhelming privilege on the confidence interval.

5.3.3 | Extendability

The scale of IBM SP2 is far smaller than that of hybrid clouds. The following experiments try to validate its extendability when the number of schedulers increases.

The system is initialized by the SP2 log–based data set with fixed execution time. There are 7 schedulers and 128 PEs. Then we increase the schedulers but not change the PEs. The 128 PEs belong to the schedulers randomly. Jobs only arrive at the initial 7 schedulers according

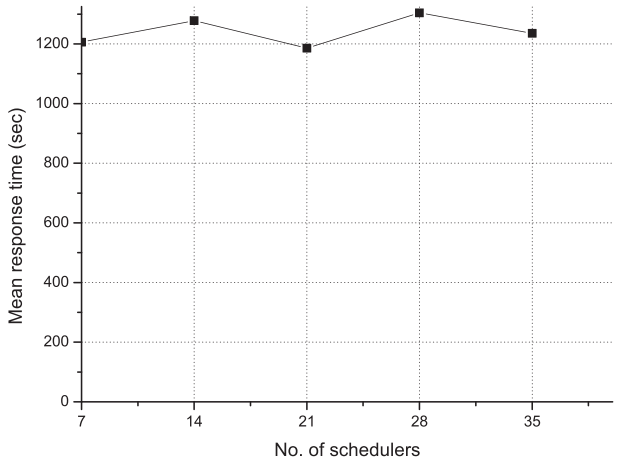

FIGURE 12 Mean response time under different numbers of schedulers

to the SP2 log. For simplicity, still assume all of the schedulers fully connected and at the same layer.

Figure 12 shows the MRTs under different numbers of schedulers. Because the number of PEs remains the same, the MRTs stay around 1210seconds no matter how many schedulers there are. As the number of schedulers increases, the forwarding path may get longer but the waiting time in each scheduler gets shorter. So the large scale of clouds does not influence the performance of our algorithm severely.

6 | RELATED WORK

Cloud computing emerges as the dramatic development of grid computing, virtualization, and web technologies. The real strength of the cloud lies in the ability to distribute the workload over multiple nodes using multiple technologies to create a high-performance, highly scalable, and available platform at a highly affordable price.[17,18] Some of the applications of cloud computing are online gaming, social networking, and data center.[19] For economic and efficiency reasons, hybrid clouds further integrate private clouds and public clouds into 1 unified environment.[20]

Millions of users share cloud resources by submitting their computing task to the cloud system. Scheduling these millions of tasks is a challenge to cloud computing.[21]

Much research has been done on task scheduling. These algorithms mainly fall into 2 categories. One is the static scheduling, which determines where and when to execute a task at compiling time. The other is the dynamic scheduling, which makes such decisions at runtime. In static scheduling,[22] knowledge on tasks and system state should be determined a priori. Because of the uncertainties, such knowledge can only be acquired at running time. So traditional static scheduling is not applicable.

In dynamic scheduling, most works try to adapt to job arrival pattern and execution dynamics by the way of workload modeling to predict system state. Smith et al.[23] proposed a run-time method to predict how long applications will wait in a queue and therefore improve scheduling performance. An online system for predicting batch-queue delay was proposed by Nurmi et al.[24] Rao and Huh [4] have proposed a probabilistic and adaptive job scheduling algorithm using system-generated predictions for grid systems. The proposed algorithm first uses system-generated job execution time estimates without actually submitting jobs to the target resource. Then, this estimation is used to predict the job scheduling feasibility on the target system. Reza and Ali[5] presented a probabilistic task scheduling method to minimize the overall MRT of the tasks submitted to the grid computing environments. A discrete time Markov chain representing the task scheduling process within the grid environment is constructed. Then a nonlinear programming problem is defined. These approaches take advantage of queueing theory and statistics to model highly dynamic variables. But our method learns those dynamics from the run-time samples without building any explicit models. Model free makes scheduling less complicated and suffers less from the error caused by models. However, models have priority of quality of service scheduling,[25] on which our method is not so straightforward. Quality of service is part of our next work.

In hybrid clouds, schedulers need to adapt to behaviors of other peers except for adaptation to those dynamics. As for collaboration, there are 2 kinds of schedulers, ie, benevolent and self-interested. Different approaches should be adopted to achieve collaboration under each case. For self-interested schedulers, which are not always willing to cooperate especially by sacrificing their profit, game theory is useful to spontaneously form a federation.[26] Subrata et al.[27] modeled the grid load-balancing problem as a noncooperative game. In 1 study,[28] Nash bargaining solution and Nash equilibrium were adopted for single-class and multiuser jobs, respectively. For benevolent schedulers, Grosu et al.[29,30] studied load balancing based on cooperative game theory. These algorithms are all static on the assumption that job arrival and execution conform to queueing theory. Actually, they may fail under non-Markovian environment.[31] The work [32] implements a dynamic scheduling for coordination. But it focuses on making up for inaccurate run-time estimates to improve the response time. It is different from ours in the following aspects. First, prediction model is its basis. Second, the metascheduler leads to a centralized approach. Instead, ours are distributed. At last, all the service providers are assumed dedicated, not their own job flows, wherein the case is simpler because it is unnecessary to mutually adapt to others' job arrival patterns and scheduling behaviors.

7 | CONCLUSION

Hybrid clouds have drawn more and more attention either in business or in academia. The public and private cloud infrastructures, which operate independently of each other, collaborate, lowering infrastructure cost and enhancing quality of experience.

Aiming at providing an approach to technically achieve effective collaboration, this paper proposes a distributed scheduling algorithm for hybrid clouds. It shortens the MRT and improves efficiency of collaboration. Once cloud providers agree to form cooperative federation, they will inform their partners the scheduler list, which is used to interact between clouds. Then the partner providers configure part or all of their schedulers to contain them as the schedulers at the upper layer.

That is, only by extending the *HS* sets in our algorithm will a federation with effective collaboration be founded.

However, when there is no such federation, whether collaboration is possible is the problem we are going to investigate next. Game theory is one way to propose an automatic negotiation mechanism that builds spontaneous cooperation when each cloud is self-interested.

ACKNOWLEDGEMENTS

The authors would like to thank the editor and anonymous reviewers for their insightful comments that considerably helped us to enhance this paper. This research was partially funded by the Key Program of National Natural Science Foundation of China (grant no. 61133005) and 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. Reaping Business Value from a Hybrid Cloud Strategy, 2015. (Available from: https://www.fujitsu.com/es/Images/HybridCloud Services-WhitePaper.pdf) [Accessed on 13 April 2015].
2. Journey to the Hybrid Cloud, white paper, 2015. (Available from: https://www.vmware.com/files/pdf/idc-hybrid-cloud-defined-white-paper.pdf) [Accessed on 1 September 2015].
3. Khan AR, Othman M, Madani SA, Khan SU. A survey of mobile cloud computing application models. *IEEE Commun. Surv. Tut.* 2014;16(1): 393–413.
4. Rao I, Huh E-N. A probabilistic and adaptive scheduling algorithm using system-generated predictions for inter-grid resource sharing. *J. Supercomput.* 2008;45(2):185–204.
5. Reza E-M, Ali M. A probabilistic task scheduling method for grid environments. *Future Gener. Comp. Syst.* 2012;28(3):513–524.
6. Omara F, Arafa M. Genetic algorithms for task scheduling problem. *J. Parallel Distrib. Comput.* 2010;70(1):13–22.
7. Umarani SG, Maheswar VU, Shanthi P, Siromoney A. Tasks scheduling using ant colony optimization. *J. Comput. Syst. Sci.* 2012;8(8): 1314–1320.
8. Jiang Y, Jiang J. Understanding social networks from a multiagent coordination perspective. *IEEE Trans. Parallel Distrib. Syst.* 2014;25(10):2743–2759.
9. García-Cuesta E, Iglesias JA. User modeling: through statistical analysis and subspace learning. *Expert Syst. Appl.* 2012;39(5):5243–5250.
10. Sayed-Mouchaweh M, Lughofer E. *Learning in Non-Stationary Environments: Methods and Applications*. Springer: New York, 2012.
11. Kaelbling L, Littman M, Moore A. Reinforcement learning: a survey. *J. Artif. Intell. Res.* 1996;4:237–285.
12. Watkins C, Dayan P. Q-learning. *Mach. Learn.* 1992;8:279–292.
13. Singh S, Jaakkola T, Jordan M. Reinforcement learning with soft state aggregation. *Adv. Neural Inf. Process. Syst.* 1995;7:261–368.
14. (Available from: http://www.cs.huji.ac.il/labs/parallel/workload/logs.html).
15. Xhafa F, Barolli L, Durresi A. Immediate mode scheduling of independent jobs in computational grids. In *Proc. the 21st International Conference on Advanced Networking and Applications*, Niagara Falls, Canada, 2007;970–977.
16. Sahu R, Chaturvedi A. Many-objective comparison of twelve grid scheduling heuristics. *Int. J. Comput. Appl.* 2011;13(6):9–17.
17. Armbrust M. Above the Clouds: A Berkeley View of Cloud Computing, Technical Report, Univ. of California, Berkeley, Feb 2009.
18. de Assuncao MD, di Costanzo A, Buyya R. Evaluating the cost-benefit of using cloud computing to extend the capacity of clusters. In *Proceedings of the 18th ACM International Symposium on High Performance Distributed Computing (HPDC)*, Munich, Germany, 2009;141–150.
19. Barolli L, Chen X, Xhafa F. Advances on cloud services and cloud computing. *Concurr. Comput.: Practice and Exper.* 2015;27(8):1985–1987.
20. Wang W-J, Chang Y-S, Lo W-T, Lee Y-K. Adaptive scheduling for parallel tasks with QoS satisfaction for hybrid cloud environments. *J. Supercomput.* 2013;66(2):783–811.
21. Mei J, Li K, Ouyang A, Li K. A profit maximization scheme with guaranteed quality of service in cloud computing. *IEEE Trans. Comput.* 2015; 64(11):3064–3078.
22. Pinel F, Dorronsoro B, Pecero JE, Bouvry P, Khan SU. A two-phase heuristic for the energy-efficient scheduling of independent tasks on computational grids. *Cluster Sci.* 2013;16(3):421–433.
23. Smith W, Taylor V, Foster I. 1999. Using run-time predictions to estimate queue wait times and improve scheduler performance, Chapter of Job Scheduling Strategies for Parallel Processing, Springer Berlin Heidelberg, 202–219.
24. Nurmi D, Brevik J, Wolski R. QBETS: queue bounds estimation from time series. In *Proceedings of Job Scheduling Strategies for Parallel Processing (JSSPP)*, Seattle, WA, USA, 2007;76–101.
25. He L, Jarvis SA, Spooner DP, Chen X, Nudd GR. Dynamic scheduling of parallel jobs with QoS demands in multiclusters and grids. In *Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing (Grid)*, Pittsburgh, USA, 2004;402–409.
26. Buyya R, Ranjan R, Calheiros RN. 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)*, Busan, Korea, 2010;13–31.
27. Subrate R, Zomaya A, Landfeldt B. Game-theoretic approach for load balancing in computational grids. *IEEE Trans. Parallel Distrib. Syst.* 2008; 19(1):66–76.
28. Penmatsa S, Chronopoulos AT. Game-theoretic static load balancing for distributed systems. *J. Parallel Distrib. Comput.* 2011;71(3): 537–555.
29. Grosu D, Chronopoulos A, Leung M. Load balancing in distributed systems: an approach using cooperative games. In *Proc. 16th International Parallel and Distributed Processing Symposium, IPDPS'02*, Ft. Lauderdale, FL, 2002;1–10.
30. Grosu D, Chronopoulos A, Leunga M. Cooperative load balancing in distributed systems. *Concurr. Comput.: Pract. Exper.* 2008;20(16): 1953–1976.
31. Pezoa J, Hayat M. Performance and reliability of non-Markovian heterogeneous distributed computing systems. *IEEE Trans. Parallel Distrib. Syst.* 2012;23(7):1288–1301.
32. Netto MAS, Buyya R. Coordinated rescheduling of Bag-of-Tasks for executions on multiple resource providers. *Concurr. Computat. Pract. Exper.* 2012;24(12):1362–1376.

How to cite this article: Xiao, Z., Liang, P., Tong, Z., Li, K., Khan, S. U., and Li, K. (2016), Self-adaptation and mutual adaptation for distributed scheduling in benevolent clouds. *Concurrency Computat.: Pract. Exper.*, doi: 10.1002/cpe.3939