Received April 6, 2021, accepted May 5, 2021, date of publication May 10, 2021, date of current version June 11, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3078773

A Self-Adapting Task Scheduling Algorithm for Container Cloud Using Learning Automata

LILU ZHU[1,2,3], KAI HUANG[2,3], YANFENG HU[2,3], AND XIANQING TAI[2]
[1]School of Information Science and Technology, University of Science and Technology of China, Hefei 230026, China
[2]Institute of Electronics, Chinese Academy of Sciences, Suzhou 215123, China
[3]Key Laboratory of Intelligent Aerospace Big Data Application Technology, Suzhou 215123, China

Corresponding author: Lilu Zhu (zhull@aircas.ac.cn)

This work was supported by the Strategic Priority Research Program of the Chinese Academy of Sciences under Grant XDA19080201.

ABSTRACT With the rapid development of cloud computing and container technology, more and more applications are deployed to the cloud, and the scale of cloud platform is expanding. Due to the large number of container instances running in the platform, complex dependency relationship, fast version iteration and other characteristics, the update of business can often cause the change of the whole cloud resource environment, which triggers the repetitive scheduling problem of related tasks and affects stability of the business. In this paper, we propose a self-adapting task scheduling algorithm (ADATSA) using learning automata to solve these problems. Firstly, we design a learning automata model and objective function for the system on task scheduling problem. Then, we realize an effective reward-penalty mechanism for scheduling actions in combination with the idle state of resources and the running state of tasks in the current environment. Meanwhile, the environment is modeled by cluster, node and task, and the probability of action selected is optimized by scheduling execution, thus enhancing the adaptability to the cloud environment of the scheduling and accelerating convergence. Finally, we construct a framework of task load monitoring with buffer queue to achieve dynamic scheduling based on priority. The experimental part verifies the effectiveness of proposed algorithm with different angles such as resource imbalance degree, resource residual degree and QoS. Compared with other learning automata scheduling models such as LAEAS, non-automata technology based algorithms such as PSOS and K8S scheduling engine, ADATSA shows the better performance of environment adaptability, resource optimization efficiency and QoS in dynamic scheduling. The theoretical analysis was consistent with the experimental results.

INDEX TERMS Container cloud, learning automata, self-adapting scheduling, reward-penalty strategy.

I. INTRODUCTION

As an important supporting technology of cloud computing, container cloud [1] has further absorbed the essence of distributed computing theory and technology such as parallel computing and grid computing. Relying on the lightweight, low resource consumption and fast start up advantages of containers, container cloud has gained rapid development and application in the internet field in recent years. Meanwhile, with the rise of container orchestration engines such as Kubernetes [2], Mesosphere [3] and Docker Swarm [4], more and more applications are virtualized and deployed to cloud, hosting the whole runtime environment and resource management of software to the container cloud platform. Container cloud technology has changed the traditional resource management of data center and improved the efficiency of business collaboration. However, how to improve comprehensive utilization of basic resources through simple and reasonable task scheduling is still a key issue of container in the cloud platform research. In particular, in the face of massive tasks access and data processing scenarios, reasonable resource allocation of task is very important, which directly affects the performance of the task and even the availability of the business.

The traditional task scheduling method of container cloud is static scheduling [5], [6]. Once the task is scheduled successfully, the container and node will maintain a strong binding relationship for a long time until the task is unloaded and deleted, and the resources occupied by the container can be recovered and reused. With the continuous updating,

The associate editor coordinating the review of this manuscript and approving it for publication was Taehong Kim.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

going online and offline of all kinds of applications, the container cloud environment becomes more stochastic and complex, it is prone to the phenomenon of extreme imbalance of node resources, which leads to the problem of business stability and even availability. In this case, static scheduling requires the operators to be involved in unloading some of the high-load tasks and then rebalancing the cluster resources. This is not only a heavy task, but also sometimes problems cannot be solved in a timely and effective manner, resulting in huge losses. Therefore, dynamic scheduling [7]–[9] appears, which releases resource pressure on the nodes with high load by dynamically migrating some runtime containers, and finally achieves global resource balance. Significantly, most of existing researches on the technology of dynamic scheduling focus on optimization of scheduling strategy to achieve the best match between tasks and resources. However, due to the differences of basic resources, user needs and operation strategies, it is difficult to achieve the ideal requirements of business initialization and operation. At the same time, for some application links with complex relationships, the change of a single service node will cause the resource reorganization of the whole link. Therefore, it is necessary to improve the perception and adaptability of tasks to different environments of container cloud. Through the continuous attempt-feedback mechanism, the relationship between task and environment can be established as soon as possible to maximize the revenue of task scheduling.

Reinforcement Learning (RL) [10] is one of the paradigms and methodologies of machine learning. It is used to describe and solve the problem, which agents achieve the maximum return or specific objective through learning strategies in the process of interaction with the environment. Learning Automata (LA) [11] is a kind of technology for reinforcement learning. It runs in probability space, is not affected by the imbalance of samples, and has good noise robustness and global optimization ability. Since the concept of learning automata was put forward, it has experienced decades of development, and the convergence speed of its algorithm has been greatly improved. The study of learning automata is not only to improve the algorithm, but also involves how to apply learning automata to solve various practical problems [12]. Research shows that learning automata is suitable for the analysis and modeling of random environment, and widely used in the fields of distributed computing, image processing, wireless sensor network and artificial intelligence. It also has relevant research results in static scheduling of container cloud tasks [30]–[33]. However, it is lack of the application research in dynamic scheduling of container cloud tasks. In fact, the self-adapting learning ability of learning automata to random environment is also suitable for whole running process of container task, that means, for the scenario of dynamic scheduling, interactive learning with running environment is realized through execution of scheduling actions, and cumulative rewards and penalties of different scheduling action probabilities are realized through feedback of scheduling results. Finally, the optimal or near optimal dynamic scheduling strategy of a certain kind of task is outputted in the current environment.

In this paper, we use a reinforcement learning method to model tasks and environment. Aiming at the environmental uncertainty of container cloud platform in the process of practical application, we introduce the learning automata theory, construct the task scheduling model through continuous interaction between task and environment, decouple strong dependence of traditional scheduling algorithm on the environment, and design a kind of portable self-adapting scheduling scheme. The main contributions of this paper are as follows:

- Based on the traditional learning automata theory, we design a Task Scheduling Oriented Learning Automaton Model (TSOLAM). and propose a model-based self-ADApting Task Scheduling Algorithm (ADATSA). The algorithm changes the dependence of traditional dynamic scheduling strategy on the infrastructure environment of container cloud, enhances the correlation between tasks and running environment through scheduling experience accumulation, and optimizes the scheduling strategy.
- We extend ADATSA algorithm from the perspective of engineering application, design an Environment-based task Load Monitoring framework (ELM), which is used for real-time monitoring of resource environment and scheduling evaluation feedback, realize an automatic scheduling learning method, and improve execution efficiency of the algorithm.
- By simulating different scheduling scenarios, the effectiveness of algorithm is proved. Compared with other earlier learning automaton models and scheduling algorithms, ADATSA has a better performance in resource utilization, resource imbalance degree and quality of service.

The rest of this paper is organized as follows. In Section II, we summarize the related work. Section III gives the related concepts and theoretical knowledge involved in this paper. In Section IV, we present task-oriented learning automaton model TSOLAM and self-adapting scheduling algorithm ADATSA in detail. Section V evaluates the proposed algorithm by real environment experiments. In Section VI, we conclude this paper and look for future work.

II. RELATED WORK

As a highly practical technology, container cloud task scheduling has been widely studied in academic and industrial fields in recent years. From the perspective of optimization objective, these studies focus on the scheduling optimization of single or multi-objective, such as optimization of resource utilization [13], scheduling efficiency [14], task completion time [15], QoS [16] and so on. From the perspective of scheduling algorithm, the scheduling optimization based on heuristic algorithm [17]–[22], mathematical model [23]–[25], automata theory [26]–[32], and other algorithms are studied. This paper summarizes the research progress of task scheduling from the perspective of algorithm classification.

The scheduling optimization based on heuristic algorithm is mainly aimed at improvement of ant colony algorithm, particle swarm algorithm and genetic algorithm, and makes them to meet requirements of the task scheduling scenarios in cloud environment. Ref. [5] proposed a multi-objective optimal scheduling method based on ant colony algorithm, which considers the utilization of node computing and storage resources, as well as the number of microservice requests and failure rate. This method uses quality evaluation function of feasible solution to ensure the effectiveness of pheromone update, and combines with multi-objective heuristic information to improve selection probability of optimal path. In Ref. [17], a multi-objective optimization scheduling method by establishing a resource cost model is proposed. This method takes manufacturing cycle and user budget cost as constraints of the optimization problem, and achieves dual objective optimization of performance and cost through ant colony algorithm. According to Ref. [18], in view of the large number of data communication and collaborative computing scenarios in application of cloud computing environment, based on particle swarm optimization algorithm, the multi-objective optimization model of resource utilization, resource imbalance and service access delay is established to achieve the comprehensive optimization of cluster resources and service performance. For the diversified, highly complex and massive task scheduling in cloud environment. Ref. [19] proposed a new adapting genetic algorithm. It can maximize the retention of excellent individuals and greatly reduce the probability of the algorithm falling into the local optimal solution by improving crossover mutation genetic operator. Ref. [20] proposed an improved multi-objective genetic algorithm based on elitist archiving and K-means method. This algorithm is used to solve the problem of complex resource allocation in multi-tenant computing environment. In addition, Ref. [21]–[23] based on bat algorithm and other methods optimized resource utilization and reduced total time of task execution. Heuristic scheduling algorithm searches optimal solution from the whole solution space, which can obtain better optimization effect, however, it takes much optimization time, and brings scheduling complexity. What's more, it is difficult to adapt to some high real-time scheduling scenarios. For example, in the network fluctuation environment, the application is prone to failure. In order to ensure continuity of the service, it is necessary to pull up the failure container quickly, which puts forward higher requirements for the real-time scheduling algorithm.

The scheduling optimization based on mathematical model, through mathematical modeling of scheduling problem, obtains the optimal solution of the scheduling problem by solving the model. Ref. [24] proposed an effective adapting scheduling algorithm by modeling the scheduling problem as integer linear programming. Ref [25] proposed a linear scheduling model for application scenarios of 5G edge computing based on resource utilization to calculate

TABLE 1. Cloud resource scheduling algorithm comparison.

References	Methodology	Advantages	Disadvantages
[5]	Microservice scheduling based on ant colony	• Multi-objective optimization	• High time complexity • Slow convergence
[18]	Scheduling approach in containerized cloud based on PSO	• Multi-objective optimization • Performance-based service scheduling	• Easy to fall into the local optimum
[19]	Scheduling of cloud service resources based on genetic algorithm	• Strong global search ability • Collaborative optimization scheduling	• High time cost • Slow convergence
[24]	Resource scheduling optimization using linear programming model	• Adaptive scheduling • Low network delay	• Single goal optimization
[29]	Job scheduling using cellular automata and ant colony	• Fast convergence • Easy to implement	• Single goal optimization
[31]	Task scheduling for cloud using learning automata	• Multi-objective optimization • Multi-factor comprehens-ive evaluation	• Slow convergence
[33]	Task scheduling in distributed systems using learning automata	• Multi-objective optimization • Fast convergence	• Easy to fall into the local optimum

metric weight, so as to avoid the scheduling applications to edge nodes with disk and bandwidth saturation. In Ref. [16], a resource scheduling algorithm based on Markov prediction model is proposed to realize task migration decision in case of node failure. Ref. [6] described the relationship among host, virtual machine, and container through UML model, established a calculation model of service reliability through mean failure time and mean failure recovery time, and realized the placement of containers to virtual machine node with the highest reliability. In Ref. [26], a utility maximization model is established, and a simplified gradient algorithm is designed to solve the Lagrangian dual problem. The simulation results show that the algorithm has good convergence and can realize hierarchical allocation of basic resources. The task scheduling method based on mathematical model often needs strict mathematical derivation, which has high computational complexity, and the scheduling model based on ideal assumption has low adaptability to environmental changes, so it is difficult to reach a consistent scheduling result.

The scheduling optimization based on automata theory mainly uses learning automata, cellular automata, and other theories to solve the task scheduling problem. In Ref. [27], bat algorithm and cellular automata theory were combined to establish the optimization objective of minimizing cost and task completion time. And ROV coding technology was applied to bat algorithm to encode and decode individual position vector and velocity vector to optimize the objective. Aiming at the problem that ant colony algorithm is easy to fall into local optimal solution, Ref. [28] improved ant colony algorithm based on cellular automata model. Through evolution mechanism of cellular automata, the optimal solution obtained by ants looking for food is reallocated, and the convergence speed of the algorithm is optimized. Ref. [29] proposed a scheduling algorithm based on sand heap cellular automata, which improves fairness of concurrent jobs by minimizing running speed of applications and dynamically scheduling and allocating resources. Ref. [30] proposed a dynamic fault-tolerant task scheduling algorithm using learning automata, which establishes a dynamic task runtime model based on normal distribution to optimize QoS and energy consumption. In Ref. [31], a new cloud task scheduling algorithm based on learning automata is proposed to solve dual objective minimization problem of energy consumption and task completion time. Ref. [32] proposed a random learning machine scheme, which is based on random learning paradigm of two-time scales to achieve dynamic load balancing in the cloud environment. In Ref. [33], for the task scheduling problem of distributed system, aiming at slow convergence of genetic algorithm solution space, learning automata is used for local search, which is superior to the existing methods based on genetic algorithm in terms of communication cost, CPU utilization and task completion time. The task scheduling based on automata theory effectively integrates the scheduling algorithm and automata theory. It has a complete theoretical basis, combined with multi-party advantages, and has a good performance in resource balanced scheduling. In particular, the scheduling based on learning automata theory has good noise robustness and global optimization ability in random environment, which is especially suitable for large-scale task scheduling optimization scenarios in relatively resource constrained environment. However, the existing task scheduling algorithms based on learning automata using RP (reward-penalty) probabilistic updating strategy (such as Ref. [31]) lack of difference in the probability reward granularity of different scheduling actions, which leads to the problems of large randomness and weak directivity when searching for the optimal solution in the solution space, so there is still a large room for improvement.

To sum up, the scheduling based on heuristic algorithm gains scheduling performance through time cost. The scheduling based on model is difficult to achieve optimal scheduling due to low environmental adaptability. And the scheduling based on automata theory performs better in efficiency and performance. Based on the theory of learning automata, we propose a dynamic task scheduling solution for container cloud. It realizes dynamic optimal scheduling of large-scale tasks and improves resource utilization efficiency of container cloud cluster.

III. RELATED CONCEPTS OF ADATSA

In order to help understand, this paper gives the related concepts and theories, including container cloud, cloud task, learning automata and so on.

- Container cloud [1]. Taking container as basic unit of resource partition and scheduling, it encapsulates the whole runtime environment of software and provides a platform for developers and system administrators to build, publish and run distributed applications.
- Cloud task. This paper refers to the container scheduling task, which encapsulates executable business program, such as web application.
- Resource Residual Degree (*RRD*) [34]. The ratio of remaining amount of resources to the total amount of resources, which indicates the utilization of system resources.
- Resource Imbalance Degree (*RID*) [35]. The degree of imbalance in the use of various resources is expressed by root mean square error of the *RRD* in cluster environment.
- Node resource priority. In this paper, the advantages and disadvantages of node resources are determined by *RRD* and *RID* of node resources. The higher the node resource priority is, the better the node resource status is.
- Steady-state time. This paper refers to the time when tasks run stably on the nodes, that is, the tasks do not migrate to other nodes during this period.
- Node stability degree. In this paper, the stability of a node is expressed by the average steady-state time of all tasks carried by the node.
- Task stability degree. In this paper, the stability of a task is expressed by the average steady-state time of the task on all its migrating nodes.
- Quality of Service (QoS) [36]. It is a kind of network security mechanism and technology used to solve problems of network delay and blocking. The goal of QoS is to provide guarantees on the ability of a network to deliver predictable results. Elements of network performance within the scope of QoS often include availability (uptime), bandwidth (throughput), latency (delay), and error rate.
- Learning automata (LA) [11], [12]. LA is a kind of algorithm in machine learning, which runs in probability space and learns the optimal value through constant interaction with unknown environment. As shown in Fig. 1, the learning automata adjusts probability distribution of each action selected according to the environment feedback (reward or penalty), and

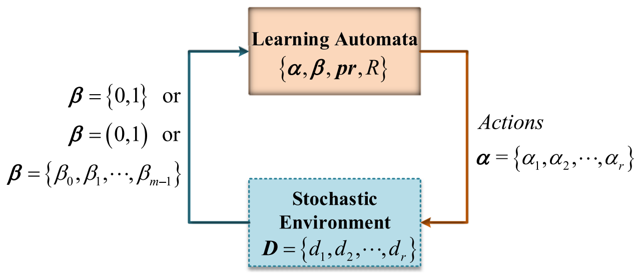

Learning Automata
$\{\boldsymbol{\alpha},\boldsymbol{\beta},\boldsymbol{pr},R\}$
$\boldsymbol{\beta}=\{0,1\}$ or
$\boldsymbol{\beta}=(0,1)$ or
$\boldsymbol{\beta}=\{\beta_0,\beta_1,\cdots,\beta_{m-1}\}$
Actions
$\boldsymbol{\alpha}=\{\alpha_1,\alpha_2,\cdots,\alpha_r\}$
Stochastic Environment
$\boldsymbol{D}=\{d_1,d_2,\cdots,d_r\}$

FIGURE 1. A learning automata and its stochastic environment.

makes probability value converge to the best action. A typical LA is defined by a four tuple $\langle \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{pr}, R\rangle$, while its environment is a three tuple $\langle \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{D}\rangle$. The definitions of related symbols are shown in Table 2. Through reinforcement learning based on environment feedback, LA constantly adjusts the probability distribution of selected actions, and finally converges to one of them. The environment feedback values are different for different environment types, in which P-type environment and Q-type environment are discrete values, and S-type environment is a continuous value between 0 and 1. The probability updating strategy mainly includes RP (reward-penalty), RI (reward-inaction) and IP (inaction-penalty). RP is the reward penalty strategy, that is, increase probability of selecting the action when it is rewarded by the environment. On the contrary, reduce probability of selecting the action when it is penalized. RI is the reward inaction strategy, that is, increase probability of selecting the action when it is rewarded by environment. on the contrary, it will not take any action. IP is the inaction penalty strategy, that is, reduce probability of selecting this action when it is penalized by environment. on the contrary, it will not take any action.

TABLE 2. Symbol definition.

Symbol	Description
$\boldsymbol{\alpha}$	action set $\{\alpha_1, \alpha_2, \cdots,\alpha_r\}$
$\boldsymbol{\beta}$	environment feedback: in S-type environment, $\boldsymbol{\beta}$ is continuous value between 0 and 1. in P-type environment, $\boldsymbol{\beta}$ is 0 or 1. in Q-type environment, $\boldsymbol{\beta}$ is several fixed values.
$\boldsymbol{pr}$	probability of action being selected $\{pr_1, pr_2, \cdots, pr_r\}$
R	probability updating strategy, mainly including RP (reward-penalty), RI (reward-inaction) and IP (inaction - penalty)
$\boldsymbol{D}$	reward probability of environment to action $\{d_1, d_2,\cdots, d_r\}$

IV. SELF-ADAPTING TASK SCHEDULING ALGORITHM

This section mainly introduces our self-adaptive scheduling algorithm ADATSA for container cloud tasks. Firstly, the task scheduling problem of container cloud is described to further clarify the problem to be solved in this paper. Secondly, a task-oriented learning automaton model TSOLAM is proposed, and the corresponding adaptive scheduling algorithm ADATSA is introduced in detail. Finally, the framework of task load monitoring named ELM is designed, and the buffer queue is constructed to realize dynamic priority scheduling

A. PROBLEM DESCRIPTION

In container cloud platform, applications need to be imaged and encapsulated before they are deployed to the cloud. Container is the smallest unit of task scheduling in the platform. The goal of task scheduling is to distribute container instances to the best node, to realize rational use of various resources. The task scheduling in container cloud can be described as the problem of mapping relationship between containers and nodes. There are N^M kinds of mapping relationship between M containers and N nodes. To find the optimal mapping is a NP-hard [37], [38] problem recognized by the industry. As mentioned above, the traditional static task scheduling method is difficult to meet requirements of users in terms of QoS and resource utilization under dynamic change of environment. The emergence of dynamic scheduling, especially the dynamic scheduling based on container technology, provides an opportunity to solve the problem. Container creates a good condition for dynamic migration of tasks with its advantages of lightweight, easy resource quota, fast start-up/shutdown and so on. Scheduler can realize the service capability's senseless handoff by pulling up a new container at the target node quickly, and recycling resources of the emigrating node after applications run stably. However, due to the large number of container cloud tasks, fast version iteration, complex business relationships, and the uncertainty of environmental factors, the update of one kind of task can often cause the change of whole cloud resource environment, which may trigger the repetitive scheduling problem of related tasks, and the unreasonable dynamic scheduling scheme may affect stability of the business. Therefore, it is necessary to improve perception ability of platform's scheduling mechanism to resource environment, and make it adapt to changing applications scenarios of cloud platform.

Learning automata (LA) runs in probability space and iterates the optimal scheduling scheme through continuous interaction with environment. It has good noise robustness and global optimization ability. Based on LA theory, we construct a learning automata model for container cloud tasks, and present a self-adapting scheduling algorithm ADATSA to solve the dynamic scheduling problem of container cloud tasks.

B. TASK SCHEDULING ORIENTED LEARNING AUTOMATON MODEL

Fig. 2 shows the structure of Task Scheduling Oriented Learning Automaton Model (TSOLAM), which is composed of four parts: Learning Automata (LA), Scheduling Actuator (SA), Container Cloud Environment (CCE) and Environment and Load Monitor (ELM). Among them, LA is the core of TSOLAM, which is defined as four tuples $\langle \boldsymbol{\alpha}^{s_i}(t), \boldsymbol{\beta}^{s_i}(t), \boldsymbol{p}^{s_i}(t), R\rangle$. A scheduling task s_i corresponds

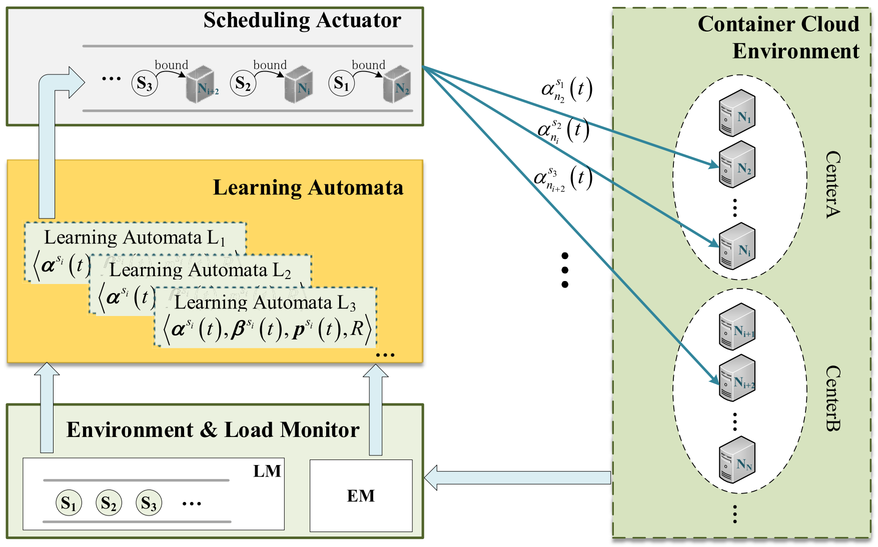


FIGURE 2. Task scheduling oriented learning automaton model.

to a LA. Through interactive learning with CCE, LA outputs the optimal scheduling scheme in current environment. SA is the executor of scheduling task. It is responsible for the implementation of the optimal scheduling scheme submitted by LA. CCE is the container cloud environment composed of multiple physical host nodes in multiple centers, which is defined as a triple $\langle \boldsymbol{\alpha}^{s_i}(t), \boldsymbol{\beta}^{s_i}(t), \boldsymbol{D} \rangle$. ELM is the monitor of environment state and task load, including Environment Monitor (EM) and Load Monitor (LM), which is used for resource and environment monitoring and scheduling evaluation feedback. TSOLAM adjusts the probability of task allocation to nodes through continuous interaction with random environment, to obtain the optimal scheduling in current environment. The related parameters are described as follows:

(1) $\boldsymbol{\alpha}^{s_i}(t)$ is a set of scheduling actions that assign task $s_i(i = 1, 2, \ldots, M)$ to node $n_j(j = 1, 2, \ldots, N)$ at time t.

(2) $\boldsymbol{\beta}^{s_i}(t)$ is possible feedback information of the environment at time t, which is the input of TSOLAM. TSOLAM uses P-type environment binary combination parameter $\{0,1\}$, 0 means that the scheduling is not conducive to system objective G, 1 is the opposite. G is described in detail below.

(3) $\boldsymbol{p}^{s_i}(t)$ is the probability set $\{p^{s_i}_{n_1}(t), p^{s_i}_{n_2}(t), \cdots, p^{s_i}_{n_N}(t)\}$ of assigning tasks s_i to node n_j at time t. TSOLAM dynamically adjusts the probability distribution $\boldsymbol{p}^{s_i}(t)$ of the selected action $\boldsymbol{\alpha}^{s_i}(t)$ according to the environment feedback $\boldsymbol{\beta}^{s_i}(t)$ (reward or penalty).

(4) R is the probability update rule of TSOLAM, including RP, RI, and IP. The mapping relationship is $p^{s_i}_{n_i}(t+1) = R\left[\alpha^{s_i}(t), \beta^{s_i}(t), p^{s_i}_{n_i}(t)\right]$. This paper adopts RP strategy, namely reward-penalty strategy. Ref. [39] first proposed an implementation method of RP strategy for energy consumption optimization of sensor scheduling in the field of wireless sensor networks, And Ref. [31] used it to solve the task scheduling problem in container cloud. In this strategy, Formula (1) is used to update the probability of favorable scheduling actions, otherwise Formula (2) is used.

$$\begin{cases} p_i(t+1) = p_i(t) + \lambda_1(1 - p_i(t)) \\ p_j(t+1) = (1 - \lambda_1)p_j(t) \quad j \neq i \end{cases} \tag{1}$$

$$\begin{cases} p_i(t+1) = (1 - \lambda_2)p_i(n) \\ p_j(t+1) = \lambda_2/(r-1) + (1 - \lambda_2)p_j(n) \quad j \neq i \end{cases} \tag{2}$$

In the above Formula, λ_1 and λ_2 are reward factor and penalty factor respectively, r is the number of actions, and $p_i(t)$ is the probability of action i being selected at time t. The reward-penalty strategy can obtain the approximate optimal solution of scheduling problems through repeated iterations. However, its probability update randomness is strong. And its probability reward granularity is lack of difference between different actions. These leads to the problems of large randomness and weak directivity in the solution space searching for the optimal solution, and then affects the convergence speed of the algorithm. According to the characteristics of container cloud task scheduling, a reward base probability $f(s_i, n_j)$ related to nodes and tasks is added to each scheduling task to enhance the optimization ability of the algorithm.

For scheduling action that is conducive to the system objective, Formula (3) is used to update the scheduling action probability.

$$\begin{cases} p^{s_i}_{n_j}(t+1) = p^{s_i}_{n_j}(t) + \lambda_1 f(s_i, n_j) \\ p^{s_i}_{n_k}(t+1) = p^{s_i}_{n_k}(t) + \lambda_1\left(f(s_i, n_k) - \frac{1}{N-1}\right) \\ 1 \leq k \leq N \;\&\; n_k \neq n_j \end{cases} \tag{3}$$

where N is the total number of nodes, λ_1 is reward factor, $f(s_i, n_j)$ is the reward base probability attached to task s_i, and $p^{s_i}_{n_j}(t)$ is the probability of task s_i scheduling to node n_j at time t. For the selected scheduling action, Formula (3) (upper) is used to update the scheduling action probability, otherwise Formula (3) (lower) is used. Formula (3) normalizes the probabilities of all scheduling actions, namely

$$\sum_{l=1}^{N} p_{n_l}^{s_i}(t+1) = 1.$$

For scheduling action that is not conducive to the system objective, Formula (4) is used to update the scheduling action probability.

$$\begin{cases} p_{n_j}^{s_i}(t+1) = p_{n_j}^{s_i}(t) - \lambda_2 p_{n_j}^{s_i}(t) \\ p_{n_k}^{s_i}(t+1) = p_{n_k}^{s_i}(t) + \lambda_2 \left(\frac{1}{N-1} - p_{n_k}^{s_i}(t)\right), \\ 1 \le k \le N \;\&\; n_k \ne n_j \end{cases} \tag{4}$$

where λ_2 is penalty factor, and other notations are the same as Formula (3). For the selected scheduling action. Formula (4) (upper) is used to update the scheduling action probability, otherwise Formula (4) (lower) is used.

The optimized reward-penalty strategy of scheduling action can make more reasonable probability reward and penalty behavior according to the latest state of resource environment, which significantly improves the adapting ability of learning automata in the dynamic scheduling scenario of container cloud tasks.

C. IMPLEMENTATION OF THE ADATSA

1) SYSTEM OBJECTIVE

As mentioned earlier, TSOLAM strengthens learning through whether environmental feedback is conducive to system objective. The probability is enhanced when the scheduling actions conducive to the system objective, otherwise, the probability is weakened. Through continuous reinforcement learning, the optimal scheduling decision is obtained. In this section, we model the system objective, and mainly involves resource imbalance degree and resource residual degree.

The imbalance degree of resource utilization directly reflects the resource allocation state of container cloud environment, so resource imbalance degree can be used as an aspect of scheduling rationality evaluation. In addition, on the premise of meeting QoS requirements, reducing resource supply and releasing redundant resources as much as possible can effectively reduce resource waste and improve resource utilization. Therefore, resource residual degree can be used as another aspect of scheduling rationality evaluation. Assuming that $\boldsymbol{R} = \{r_1, r_2, \ldots, r_K\}$ is the set of node resources, $r_k (1 \le k \le K)$ represents the k-th resource of the node, where K is the number of resource types. In addition, defining that $E(n_j, r_k)$ and $U(n_j, r_k)$ represent the quota and usage of resource r_k on node n_j respectively, then node resource residual degree and resource imbalance degree can be expressed by Formulas (6) and (7) respectively, and the system objective can be expressed by Formula (8).

$$S\left(n_j, r_k\right) = \frac{E(n_j, r_k) - U\left(n_j, r_k\right)}{E(n_j, r_k)} \tag{5}$$

Algorithm 1 SO

INPUT: node list $\boldsymbol{L}$, node resource collection $\boldsymbol{R}$
OUTPUT: system target value G

```
1:  for each node $n_j$ in $L$ do
2:      for each resource $r_k$ in $R$ do
3:          calculate $E(n_j, r_k) - U(n_j, r_k)$
4:          calculate $S(n_j, r_k)$ using Formula (5)
5:      end for
6:      calculate node resource residual degree $S(n_j)$
        using Formular (6)
7:      for each resource $r_k$ in $R$ do
8:          cumulative $(S(n_j, r_k) - S(n_j))^2$
9:      end for
10:     calculate node resource imbalance degree $L(n_j)$
        using Formula (7)
11:     cumulative $(1 - S(n_j)) + L(n_j)$
12: end for
13: compute system objective $G$ using Formula (8) and
    return
```

$$S\left(n_j\right) = \frac{1}{K}\sum_{k=1}^{K} S\left(n_j, r_k\right) \tag{6}$$

$$L\left(n_j\right) = \sqrt{\sum_{k=1}^{K} \left(S\left(n_j, r_k\right) - S\left(n_j\right)\right)^2 \Big/ K} \tag{7}$$

$$G = \sum_{j=1}^{N} \left(1 - S\left(n_j\right) + L\left(n_j\right)\right) \Big/ 2 \tag{8}$$

Node resource residual degree $S(n_j)$ is defined as the mean value of various resource residual degrees $S(n_j, r_k)$ on the node, obviously $0 \le S(n_j) \le 1$. Node resource imbalance degree $L(n_j)$ is expressed as the root mean square error of $S(n_j, r_k)$, here $L(n_j)$ is normalized. System objective G is defined as the sum of resource residual degree and imbalance degree of all nodes in container cloud environment. The smaller the G, the more reasonable the allocation of resources in container cloud environment is, and vice versa.

The system objective G is used to evaluate the rationality of task scheduling and is run in environment monitor EM. Algorithm SO gives the calculation method of G in current environment. Among them, line 1 loops through all nodes. Lines 2-5 calculate the residual degree $S(n_j, r_k)$ of all kinds of resources on node n_j. Line 6 calculates the average residual degree $S(n_j)$ of node resources. Lines 7-10 calculate the node resource imbalance degree. Line 11 calculates the cumulative sum of resource residual degree and imbalance degree on nodes. Line 12 ends the node loop. Line 13 calculates system objective.

2) REWARD BASE PROBABILITY

The correct updating of scheduling action probability is the key to reinforcement learning of ADATSA. The container cloud environment is an unstable random environment, so the probability of reward for different scheduling actions should be different. In addition, the probability of reward for the same scheduling action should be different with the change of resource environment. In this paper, a reward base probability associated with resource and environment is used for each task to update the reward probability of scheduling action. In this way, the adaptability to environment of learning automata can be enhanced.

Mainly considering the priority of resources, the stability of nodes and tasks, we model the reward base probability from two levels of nodes and tasks in this section. Through analysis of node resource usage and task state transfer topology, the reward probability of scheduling action is quantified.

(1) Node resource priority

Node resource priority is expressed as a linear combination of node resource residual degree and node resource imbalance degree, as shown in Formula (9)

$$O\left(n_j\right) = \left(1 - L\left(n_j\right) + S\left(n_j\right)\right)/2, 0 \leq O\left(n_j\right) \leq 1 \tag{9}$$

where the definitions of $S(n_j)$ and $L(n_j)$ are shown in Formulas (6) and (7). The larger the $O(n_j)$, the better the state of node resources is, and the more conducive to stable operation of tasks.

(2) Node stability degree

Node stability degree is the relative stability of node in a cluster environment, which can be measured by the average steady-state time of all tasks on it. The running status of tasks on nodes can reflect the stability of nodes to a certain extent. We think that the longer the average stable running time of all tasks on the node, the higher the stability of the node is.

The triple $q_{i,j,t} = (s_i, n_j, t)$ is defined as the position state of task s_i during the t-th transition, which indicates that s_i is scheduled to node n_j during the t-th transition. Fig. 3 shows the topology of task state transition, showing the migration path between tasks. Define $T(q_{i,j,t}) = T(s_i, n_j, t)$ as the stable running time of s_i in state $q_{i,j,t}$, then the node stability degree can be expressed by Formula (10).

$$B\left(n_j\right) = \frac{\overline{T}\left(n_j\right)}{\sum\limits_{k=1}^{N} \overline{T}\left(n_k\right)}, 0 \leq B\left(n_j\right) \leq 1 \tag{10}$$

$$\overline{T}\left(n_j\right) = \frac{1}{\left|\boldsymbol{IN}\left(n_j\right)\right|} \sum_{s_i \in IN\left(n_j\right)} \left(\frac{1}{\left|\boldsymbol{Q}_{n_j}^{s_i}\right|} \sum_{q \in \boldsymbol{Q}_{n_j}^{s_i}} T\left(q\right) \right) \tag{11}$$

where $\boldsymbol{Q}_{n_j}^{s_i} = \left\{q_{i,j,1}, q_{i,j,2}, \cdots\right\}$ is the collection of historical transition states of s_i on node n_j, $\left|\boldsymbol{Q}_{n_j}^{s_i}\right|$ is the length of the collection, $\boldsymbol{IN}\left(n_j\right)$ is the collection of migrating tasks on node n_j, and $\left|\boldsymbol{IN}\left(n_j\right)\right|$ is the length of collection $\boldsymbol{IN}$. For example, in Fig. 3, $\boldsymbol{IN}\left(n_1\right) = \{s_1, s_2, s_3\}$, then $|\boldsymbol{IN}\left(n_1\right)| = 3$. $\overline{T}\left(n_j\right)$ is the average steady-state time of all tasks on node n_j. $B(n_j)$ is normalized result of $\overline{T}\left(n_j\right)$.

(3) Task stability degree

Task stability is relative to nodes. By analyzing the topology of task state transition, we can see that the less the number

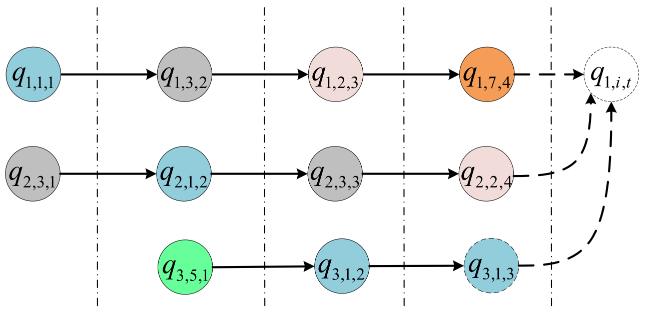


FIGURE 3. **Topology of task state transition.**

of task state transition, and the longer the average stable running time of task on all transition nodes, then the more reliable the task is. Task stability degree can be expressed by Formula (12).

$$H\left(s_i\right) = \left(1 - \frac{\left|\boldsymbol{Q}^{s_i}\right|}{\sum\limits_{i=1}^{M} \left|\boldsymbol{Q}^{s_i}\right|}\right) \frac{\overline{T}\left(s_i\right)}{\sum\limits_{i=1}^{M} \overline{T}\left(s_i\right)} \tag{12}$$

$$\overline{T}\left(s_i\right) = \frac{1}{\left|\boldsymbol{Q}^{s_i}\right|} \sum_{q \in \boldsymbol{Q}^{s_i}} T\left(q\right) \tag{13}$$

where $\boldsymbol{Q}^{s_i}$ is historical collection of tasks s_i transition states, M is the number of tasks, and $\left|\boldsymbol{Q}^{s_i}\right|$ is the length of collection $\boldsymbol{Q}^{s_i}$. Obviously $0 \leq H\left(s_i\right) \leq 1$, The larger $H(s_i)$ is, the more stable the task s_i is.

To sum up, the reward base probability $f(s_i, n_j)$ can be expressed as the weighted average of node priority, node stability degree and task stability degree, as shown in Formula (14).

$$f\left(s_i, n_j\right) = \alpha_{ij} O\left(n_j\right) + \beta_{ij} B\left(n_j\right) + \gamma_{ij} H\left(s_i\right) \tag{14}$$

where, α_{ij}, β_{ij}, γ_{ij} is the combination coefficient, which is used to adjust the proportion of the three, and $\alpha_{ij} + \beta_{ij} + \gamma_{ij} = 1$.

The RBP algorithm gives the calculation method of $f\left(s_i, n_j\right)$ in the current environment. Lines 1-4 calculate node resource priority of the current scheduling node n_j. Line 5 calculates migration task collection. Lines 6-15 calculate cumulative sum of steady-state time of all tasks on node n_j and the cumulative sum of steady-state time of task s_i on all nodes. Lines 16-18 calculate the node stability degree. Lines 19-20 calculate task stability degree. Line 21 calculates reward base probability and return.

3) ADATSA ALGORITHM

The input of ADATSA scheduling algorithm mainly includes the list of tasks to be scheduled $\boldsymbol{D} = \{s_1, s_2, \ldots, s_M\}$ and node list $\boldsymbol{L} = \{n_1, n_2, \ldots, n_N\}$. The output of the algorithm is the optimal scheduling in current environment. The running flow of ADATSA is shown in Fig. 4. The pseudo code is shown in Algorithm 3, which includes the following steps:

Algorithm 2 RBP

INPUT: node list $\boldsymbol{L}$, node resource collection $\boldsymbol{R}$, target task s_I and target node n_J, task history transition state collection $\boldsymbol{Q}$, combination coefficient α_{ij}, β_{ij}, γ_{ij}

OUTPUT: reward base probability $f(s_I, n_J)$

```
1:  calculate the resource utilization $S(n_J, r_k)$ of each resource on $n_J$ using Formula (5)
2:  calculate the node resource utilization $S(n_J)$ using Formula (6)
3:  calculate the resource imbalance degree $L(n_J)$ using Formula (7)
4:  calculate node resource priority $O(n_J)$ using Formula (9)
5:  calculate migration task collection $IN(n_J)$ on $n_J$
6:  for each task $s_i$ in $IN(n_J)$ do
7:    for each state $q$ in $\boldsymbol{Q}^{s_i}$ do
8:      if ($q_{.node}==n_J$) do
9:        cumulative $T(q)$ as $SUM1$
10:     end if
11:     if ($q_{.task} = s_I$) do
12:       cumulative $T(q)$ as $SUM2$
13:     end if
14:   end for
15: end for
16: calculate the size of $IN(n_J)$
17: calculate the node average steady-state time $\overline{T}(n_J)$ by $SUM1$ using Formula (11)
18: calculate node stability degree $B(n_J)$ using Formula (10)
19: calculate task average steady-state time $\overline{T}(s_I)$ by $SUM2$ using Formula (13)
20: calculate task stability degree $H(s_I)$ using Formula (12)
21: calculate reward base probability $f(s_I, n_J)$ using Formula (14) and return
```

Step 1: Initialization of algorithm parameters: including the initialization of reward factor λ_1, penalty factor λ_2, scheduling action probability $p_{n_j}^{s_i}$ and so on. λ_1 and λ_2 are usually initialized to be any value between 0.2~0.4, which is determined according to the actual circumstances of resources and environment. $p_{n_j}^{s_i}$ $(1 \le i \le M, 1 \le j \le N)$ is initialized as the reward base probability, that is, $p_{n_j}^{s_i} = f\left(s_i, n_j\right)$. Corresponding to pseudo code line 1 - line 2.

Step 2: Generation of scheduling scheme: LA generates the optimal scheduling scheme for current environment according to the latest scheduling action probability. For initial scheduling, the scheduling task is initial task queue $\boldsymbol{D}$. For dynamic scheduling, the scheduling task is a subset of $\boldsymbol{D}$, which is composed of high load tasks filtered by LM according to the load of nodes and tasks in current environment. The scheduling scheme is the mapping relationship between scheduling task and the optimal scheduling node.

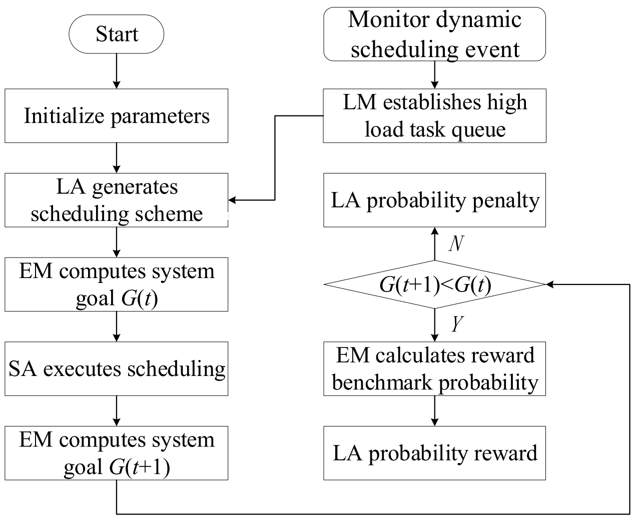

Start
Monitor dynamic scheduling event
Initialize parameters
LM establishes high load task queue
LA generates scheduling scheme
LA probability penalty
N
EM computes system goal G(t)
G(t+1)<G(t)
Y
SA executes scheduling
EM calculates reward benchmark probability
EM computes system goal G(t+1)
LA probability reward

FIGURE 4. The flow of the proposed ADATSA.

Corresponding to pseudo code line 3 - line 5. Among them, line 3 waits for the dynamic scheduling signal to be triggered. Line 4 establishes a high load task queue based on load of the cluster environment. Line 5 generates the optimal scheduling scheme according to the probability of scheduling action.

Step 3: Execution of scheduling scheme: SA is responsible for specific execution of the scheduling scheme, that is, according to the binding relationship between task and node, the target container is pulled up on the target node. Buffer queue is set in SA to relieve the peak pressure in high concurrency scheduling scenario. Corresponding to pseudo code line 7.

Step 4: Scheduling evaluation and probability update: LA requests EM to calculate the prescheduling system objective $G(t)$ (Algorithm 1) before requesting SA to execute the scheduling scheme. And then requests EM to calculate the post scheduling system objective $G(t + 1)$ after receiving the successful feedback of SA scheduling execution, and compares the two system objectives. if $G(t + 1) < G(t)$, LA thinks that the scheduling is beneficial to the system environment, requests EM to calculate the reward base probability (Algorithm 2), and uses Formula (3) to update the scheduling action probability. Otherwise, LA thinks that the scheduling is unfavorable to the system environment, and uses Formula (4) to update the scheduling action probability. Corresponding to pseudo code line 6 - line 14. Among them, lines 6 and 8 calculate the system objective before and after scheduling. Lines 9-11 reward scheduling actions that benefit the environment. Lines 12-14 penalize scheduling actions that are not conducive to the environment.

Step 5: Trigger of dynamic scheduling: This paper provides five trigger modes of dynamic scheduling, including node threshold trigger, timing trigger, etc. see Part 4) of this section for details. LM monitors triggering events of dynamic scheduling. When the triggering event is coming, a high load task queue is established according to the node and task load in current environment, and LA is requested to generate

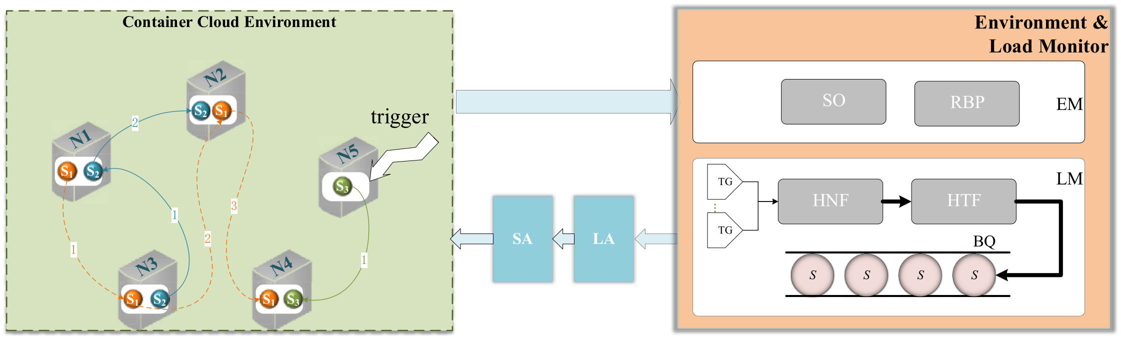


FIGURE 5. The framework of environment and load monitoring.

Algorithm 3 ADATSA

INPUT: task list $\boldsymbol{D}$, node list $\boldsymbol{L}$, reward factor λ_1, penalty factor λ_2

OUTPUT: optimal scheduling

```
1:  initialize parameters $\lambda_1, \lambda_2$;
2:  initialize all action probabilities $p_{n_j}^{s_i} = f(s_i, n_j)$
    and perform initial scheduling;
3:  waiting to trigger dynamic scheduling
4:  LM establishes high load task queue
5:  LA generate scheduling scheme according to $p_{n_j}^{s_i}$
6:  EM computes system objective $G(t)$ by Algorithm 1
7:  SA execution scheduling scheme
8:  EM computes system objective $G(t+1)$ by
    Algorithm 1
9:  if $G(t+1) < G(t)$ do
10:     EM compute reward benchmark probability
        $f(s_i, n_j)$ of all selected actions by Algorithm 2
11:     LA reward the selected actions of all tasks using
        Formula (3)
12: else
13:     LA penalize the selected actions of all tasks using
        Formula (4)
14: end if
15: waiting to trigger the next dynamic scheduling
```

optimization scheme for rescheduling. Corresponding to pseudo code line 3.

ADATSA continuously optimizes the probability of the scheduled action selected through dynamic scheduling execution, thereby enhancing the adaptability of the scheduling mechanism to resource and environment. Through a certain number of iterative learning, the optimal or near optimal scheduling in the current environment can be realized. In addition, the introduction of the reward base probability enhances the ability of the algorithm to optimize the global optimization and realizes the fast convergence of the resource utilization state.

The time consumption of ADATSA mainly lies in the calculation of system objective by Algorithm 1 and reward benchmark probability by algorithm 2. Suppose that the number of nodes in the cluster is N, the resource category is K, the average number of inbound tasks on nodes is $\overline{M}$, and the average number of state transitions of tasks is $\overline{S}$. Then the time complexity of Algorithm 1 is O $(N \times K)$, it mainly lies in the dual for loop to calculate the node resource residual and imbalance degree. The time complexity of Algorithm 2 is O $(\overline{M} \times \overline{S})$, it mainly lies in the dual for loop to calculate the steady-state time of tasks. So, the time complexity of ADATSA is O $(N \times K)$+ O $(\overline{M} \times \overline{S})$. Because line 1 for loop in Algorithm 1 and line 6 for loop in Algorithm 2 can be calculated in parallel, the time complexity of the whole algorithm is not high. The proposed ADATSA algorithm can fully meet the needs of large-scale task scheduling.

4) FRAMEWORK OF ENVIRONMENT AND LOAD MONITORING

As mentioned above, the main functions of ELM in TSOLAM are scheduling evaluation and load monitoring for nodes, which are described in detail in this section. Fig. 5 shows the framework of ELM proposed in this paper, which is composed of Environment Monitor (EM) and Load Monitor (LM). EM is composed of System Objective calculation module (SO) and Reward Base Probability calculation module (RBP). LM consists of High load Node Filter (HNF), High load Task Filter (HTF) and Buffer Queue (BQ). The SO module of EM implements Algorithm 1, its function is to evaluate the state of environment before and after task scheduling. RBP module implements Algorithm 2, which is used to calculate the reward base probability. The EM implementation feeds back the results of SO and RBP to LA, which supports the reinforcement learning of LA. LM monitors the resource load of nodes and tasks, and presses the tasks with high load that meet certain filtering conditions into the BQ, providing LA with dynamic scheduling candidate task sources. The following mainly introduces LM in detail.

The dynamic scheduling of tasks needs to meet certain trigger conditions. This paper summarizes several feasible trigger methods in container cloud environment: (1) cluster node scaling: when cluster nodes are added or deleted, trigger tasks to migrate dynamically. (2) Node threshold: when the utilization rate of some resources of cluster nodes reaches the threshold (for example, the CPU utilization rate exceeds 90%), trigger tasks to migrate dynamically. (3) Timing mechanism: regularly trigger tasks for dynamic migration. (4) Task or node exception: task exception or node failure triggers dynamic migration of tasks. (5) External control command: external control actively triggers the task for dynamic migration.

When the trigger event arrives, it is not desirable to migrate all tasks. For example, when the node resource utilization rate reaches the threshold, it only wants to transfer some tasks with high resource consumption to nodes with relatively abundant resources, thereby reducing node pressure and improving service performance. The key of task dynamic migration needs to solve two problems: one is what tasks to be migrated, and the other is how to transfer tasks. ADATSA algorithm has solved the problem of how to migrate tasks. Here, LM is used to solve the problem of which tasks to be migrated. Specifically, LM filters the task to be migrated through the following steps:

(1) Filter out applications that cannot be rescheduled. The application with state and state information stored in the local node cannot be rescheduled, because the state information will be lost when the application is rescheduled to other nodes. When the business is sensitive to data, rescheduling of the application is not allowed. Therefore, non-reschedulable applications need to be identified in the cloud platform. For example, the non-reschedulable POD restart policy in K8S cloud platform is usually set to ''never'', so it can be filtered according to the restart policy.

(2) The remaining tasks are prioritized by workload. Define $NRU(n_j)$ and $SRU(s_i)$ as the resource utilization rate of n_j and task s_i respectively. The calculation method is as follows: Formulas (17), (19). Among them, $U(n_j, r_k)$ and $P(n_j, r_k)$ are the resource usage and quota of node n_j respectively; $U(s_i, n_j, r_k)$ and $P(s_i, n_j, r_k)$ are the resource usage and requests of task s_i on node n_j respectively. The scoring mechanism shown in formula (15) is established to score the resource load of nodes and tasks comprehensively. The migration tasks are prioritized according to the score from large to small. Among them, $NSC(n_j)$ and $SSC(s_i)$ represent the resource load score of node n_j and task s_i respectively. And $NSC(n_j)$ indicates that the resource load of nodes exceeds the average level of cluster resource load. Similarly, $SSC(s_i) > 0$ indicates that the task resource load exceeds the average level of cluster tasks.

$$SC = NSC\left(n_j\right) + SSC\left(s_i\right), NSC\left(n_j\right) > 0\&SSC\left(s_i\right) > 0 \tag{15}$$

$$NSC\left(n_j\right) = \ln\left(NRU\left(n_i\right) \Big/ \frac{1}{N}\sum_{i=1}^{N} NRU\left(n_i\right)\right) \tag{16}$$

$$NRU\left(n_j\right) = \frac{1}{K}\sum_{k=1}^{K}\frac{U\left(n_j, r_k\right)}{P\left(n_j, r_k\right)} \tag{17}$$

$$SSC\left(s_i\right) = \ln\left(SRU\left(s_i\right) \Big/ \frac{1}{M}\sum_{i=1}^{M} SRU\left(s_i\right)\right) \tag{18}$$

$$SRU\left(s_i\right) = \frac{1}{K}\sum_{k=1}^{K}\frac{U\left(s_i, r_k\right)}{P\left(s_i, r_k\right)} \tag{19}$$

As shown in Fig. 5, HNF filters out the nodes whose load higher than the average resource load level of cluster nodes by Formula (16), namely $NSC\left(n_j\right) > 0$. And it passes the results to HTF. HTF further filters out the tasks whose load is higher than the average load level of cluster tasks on the node. It uses Formula (15) to score and sorts tasks according to the score. After sorting, the tasks with high load are pushed into the buffer queue (BQ), waiting for secondary scheduling. LA generates the optimal scheduling scheme for the secondary scheduling task and entrusts SA to execute it. Through a certain number of dynamic scheduling, the cluster resources are finally stable to a relatively balanced level, to improve the global resource utilization and quality of service (QoS).

V. EXPERIMENTAL SIMULATION AND ANALYSIS

In this section, we design experiments to verify the performance of the proposed ADATSA algorithm in task static and dynamic scheduling of container cloud, and compare with other learning automata based scheduling algorithms, non-automata based scheduling algorithms, and K8S scheduling engine.

A. EXPERIMENT SETTING

In the experimental environment, Kubernetes (v1.16.2) cluster was built on 53 servers with the same specifications as the experimental platform, including 3 Master nodes and 50 Slave nodes. The resource information of cluster is shown in Table 3. In addition, we use a self-developed container cloud management platform as an application deployment tool and Apache JMeter (v5.4.0) as the QoS testing tool. In terms of experimental data, tasks are divided into five types: CPU oriented (type A), memory oriented (type B), disk oriented (type C), bandwidth oriented (type D), and resource non-oriented (type E). Where resource non-oriented applications are those in which the resource requirements of the applications are balanced and there is no particular preference for resources. Task partition and resource requirement of task are shown in Table 4. Some applications provide HTTP access interfaces for QoS testing. Due to the lack of applications, we used repeated deployments to simulate large-scale application deployment. The initial number of applications used in the experiment was 100, with 20 for each type of application.

TABLE 3. 3 node resource.

Role	CPU (*core*)	Mem (*GB*)	Disk (*GB*)	Band (*Gbps*)	Amounts
Master	128	256	1000	10	3
Slave	64	128	1000	10	50

TABLE 4. Resource requirement of task.

Index	Type	CPU (*core*)	Mem (*MB*)	Disk (*GB*)	Band (*Gbps*)	HTTP API
s_1	$type_A$	1.0	512	1.0	0.05	no
s_2	$type_B$	0.5	2048	1.5	0.10	yes
s_3	$type_A$	2.0	1024	1.0	0.10	no
s_4	$type_D$	0.5	512	1.0	0.20	no
s_5	$type_C$	0.5	1024	2.0	0.15	no
…	…	…	…	…	…	…

In the evaluation of cluster resource scheduling, *RID* (resource imbalance degree) [37] and *RRD* (resource residual degree) [36] are two widely used metrics. *RID* can reflect the balance of cluster resource utilization, while *RRD* reflects the comprehensive utilization of resources. On this basis, this paper increases *RT* (response time) and *TH* (throughput) for service quality evaluation. Details of relevant metrics are as follows:

$$RID = \sum\nolimits_{j=1}^{N} L\left(n_j\right)/N \tag{20}$$

where $L(n_j)$ is shown in Formula (7). The smaller the *RID* is, the more balanced the cluster resource utilization is, and vice versa.

$$RRD = \sum\nolimits_{j=1}^{N} S\left(n_j\right)/N \tag{21}$$

where $S(n_j)$ is shown in Formula (6). The larger the *RRD* is, the more applications the cluster can deploy and the higher the cluster resource utilization is.

$$RT = \frac{1}{M}\sum\nolimits_{i=1}^{M} RT\left(s_i\right) \tag{22}$$

where $RT(s_i)$ is the response delay of application s_i (usually Web service application) under certain concurrent requests, and M is the number of applications. The average response time of a certain number of applications in cluster environment can be used as an aspect of cluster QoS evaluation.

$$TH = \frac{1}{M}\sum\nolimits_{i=1}^{M} TH\left(s_i\right) \tag{23}$$

$$TH\left(s_i\right) = \frac{N_{req}\left(s_i\right)}{T_{end}\left(s_i\right) - T_{start}\left(s_i\right)} \tag{24}$$

where $N_{req}\left(s_i\right)$ is the total number of requests for application s_i, $T_{start}\left(s_i\right)$ is start time of the test, $T_{end}\left(s_i\right)$ is end time of the test, and $TH\left(s_i\right)$ is the throughput. The average throughput of a certain number of applications can be used as another aspect of cluster QoS evaluation.

B. EXPERIMENTTAL RESULTS

In experiment A, static task scheduling is simulated, and the dynamic adjustment switch of ADATSA is closed. The impact of the number of tasks on static scheduling performance of ADATSA with that of LAEAS [31], PSOS [18] and K8S is compared. The motivation of choosing these comparison algorithms is that LAEAS is a representative scheduling algorithm using learning automata, PSOS is an excellent heuristic scheduling algorithm using non-automata based technique, and Kubernetes (K8S) platform has a strong ability of task scheduling. The experimental results are shown in Fig. 6 (a) and (b), in which $*$-S represents the static scheduling curve of algorithm $*$ (ADATSA, LAEAS, PSOS, K8S). It can be seen that the performance of ADATSA and K8S in static scheduling is close, better than LAEAS, and slightly less than PSOS. From the *RID* curve, all algorithms show a trend of rapid decline at first and then stable convergence. Although ADATSA has certain advantages when the number of tasks is between 1.3×10^3 and 2.9×10^3, with the increasing number of tasks, ADATSA converge to almost the same level as PSOS and K8S, even PSOS and K8S convergence more smoothly. From the *RRD* curve, ADATSA decreases more smoothly than K8S, but it still resides under PSOS and K8S, which indicate that the resource utilization rate of PSOS and K8S is better than that of ADATSA. This is because PSOS adopts scheduling technology using particle swarm optimization, the optimal scheduling is achieved through multiple iterations, and the scheduling performance is obtained through time. K8S framework has a complete set of basic resource optimization scheduling scheme, and achieves the optimal convergence of resource state through two stages of predicate and priority. ADATSA still has a certain gap in static scheduling compared with PSOS and K8S scheduling algorithm.

In experiment B, dynamic task scheduling scenario is simulated, and the dynamic scheduling mode of ADATSA is set as node threshold trigger. When the resource utilization rate of nodes exceeds 80%, tasks with high load were be migration automatically. Since LAEAS and PSOS can't be directly applied to the dynamic scheduling scenario in this paper, we modify them. For LAEAS, we simulate its random probability update mode, that is, we use its probability update method (Formula (1), (2)) to replace the probability update method of ADATSA (Formula (3), (4)), and other places are consistent with ADATSA. For PSOS, we regard its dynamic scheduling as static rescheduling, and dynamic scheduling performs the same scheduling process as static scheduling. In addition, because K8S has no special dynamic scheduling mechanism, we use its POD eviction function to replace (resource threshold is set to 80%). We compare the impact of the number of tasks on the dynamic scheduling performance of ADATSA and 3 comparison algorithms. The experimental results are shown in Fig. 6 (c) and (d), in which $*$-D represents the dynamic scheduling curve of algorithm $*$ (ADATSA, LAEAS, PSOS, K8S). From the convergence results of dynamic scheduling, ADATSA is better than all comparison algorithms on *RID* and *RRD*. and the performance order is ADATSA $>$ LAEAS $>$ PSOS $>$ K8S. From the *RID* curve, after the number of tasks exceeds 2.5×10^3, the *RID* metric of ADATSA continues to decline and finally converges to about 0.065. Compared with the initial state of the cluster (the number of tasks is 0), the performance of *RID* is improved by about 59.61%. Compared with the convergence result of its static scheduling

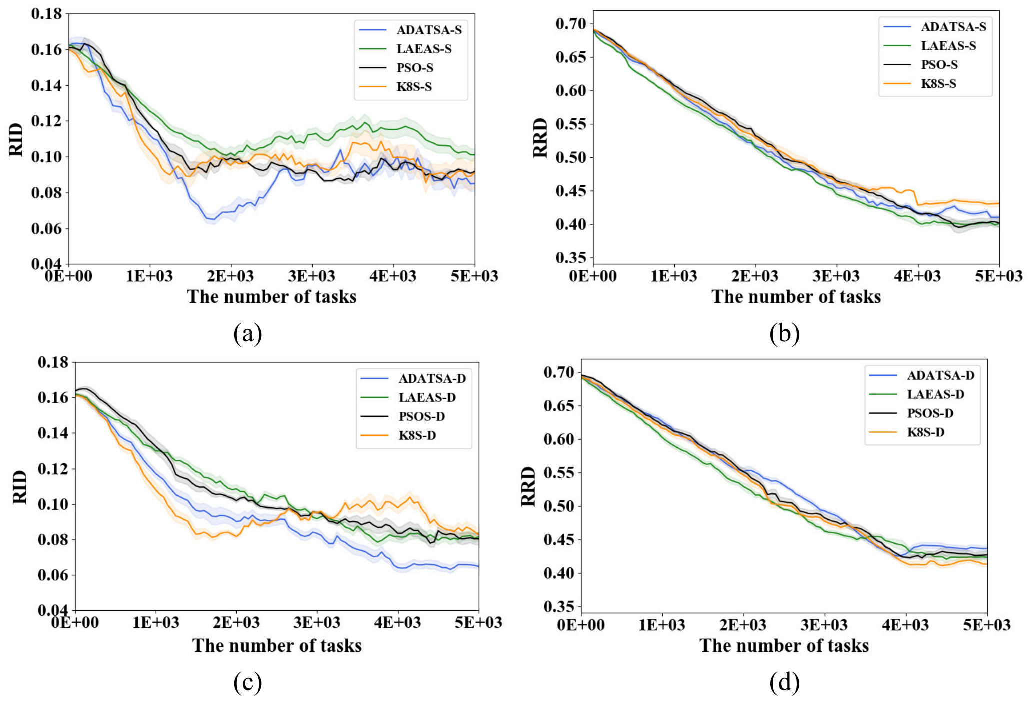


FIGURE 6. Impact of the number of tasks on static(-S) and dynamic(-D) task scheduling.

(Fig. 6 (a) ADATSA-S curve), *RID* performance is improved by about 24.01%. Compared with the convergence results of LAEAS, PSOS and K8S dynamic scheduling, the *RID* performance is improved by 19.56%, 20.17%, and 23.24% respectively. In addition, the *RRD* curve of ADATSA in Fig. 6 (d) is generally above comparison algorithms, which shows better resource utilization efficiency. This is mainly due to the effective adaptive learning ability of ADATSA, through the scheduling feedback results to adjust the scheduling behavior continuously, accumulate scheduling experience, and finally achieve a better scheduling effect. However, PSOS and K8S simply perform task scheduling, lack of reinforcement learning process for resource environment. Their dynamic scheduling process is equivalent to the repetition of static scheduling process, so the improvement of resource environment state is not particularly obvious.

In experiment C, we fix the number of tasks and verify the effect of time on the convergence performance of ADATSA and comparison algorithms. Randomly select five types of applications, create 1×10^3, 2×10^3, 5×10^3 tasks respectively, and record the resource usage every 30s. The experimental results are shown in Fig. 7 (a) to (f), in which $*$-S-1, $*$-S-2 and $*$-S-5 correspond to the static scheduling curves of algorithm $*$ (ADATSA, LAEAS, PSOS, K8S) with task numbers of 1×10^3, 2×10^3 and 5×10^3 respectively. We can see that when the number of tasks is small, the resources converge speed of ADATSA is lower than that of PSOS and K8S on *RID* and *RRD*, but significantly better than that of LAEAS. However, when the number of tasks is large, the convergence performance of ADATSA is improved to a certain extent. This is mainly because ADATSA uses the reward base probability $f(s_i, n_j)$ to initialize the scheduling action probability. $f(s_i, n_j)$ fully considers the current resource imbalance degree, resource residual degree and information of nodes, and prioritizes node resources to achieve resource priority scheduling. LAEAS algorithm uses equal probability to initialize the probability of scheduling action selection, which is random and affects the convergence performance of static scheduling.

In experiment D, we add a dynamic scheduling mechanism based on experiment C, set dynamic scheduling mode of ADATSA as timing trigger, and dynamically migrated high load tasks every 10 minutes. The impact of time on the convergence performance of ADATSA and its comparison algorithms is studied. The experimental results are shown in Fig. 7 (g) to (l), in which $*$-D-1, $*$-D-2 and $*$-D-5 correspond to the dynamic scheduling curves of algorithm $*$ (ADATSA, LAEAS, PSOS, K8S) with task numbers of 1×10^3, 2×10^3 and 5×10^3 respectively. We can see that ADATSA and comparison algorithms dynamically migrate and adjust tasks according to the load of the current resource environment, and achieve different degrees of resource convergence improvement. However, ADATSA is superior to all comparison algorithms in terms of convergence speed and final convergence result. In the dynamic scheduling scenario with 1×10^3 tasks, the *RID* performance improvement of LAEAS is weak, and there is a small divergence trend. The *RID* of PSOS is almost the same as that of K8S with medium performance. However, the *RID* curve of ADATSA decreases rapidly and converges, and it is stable at about 0.14

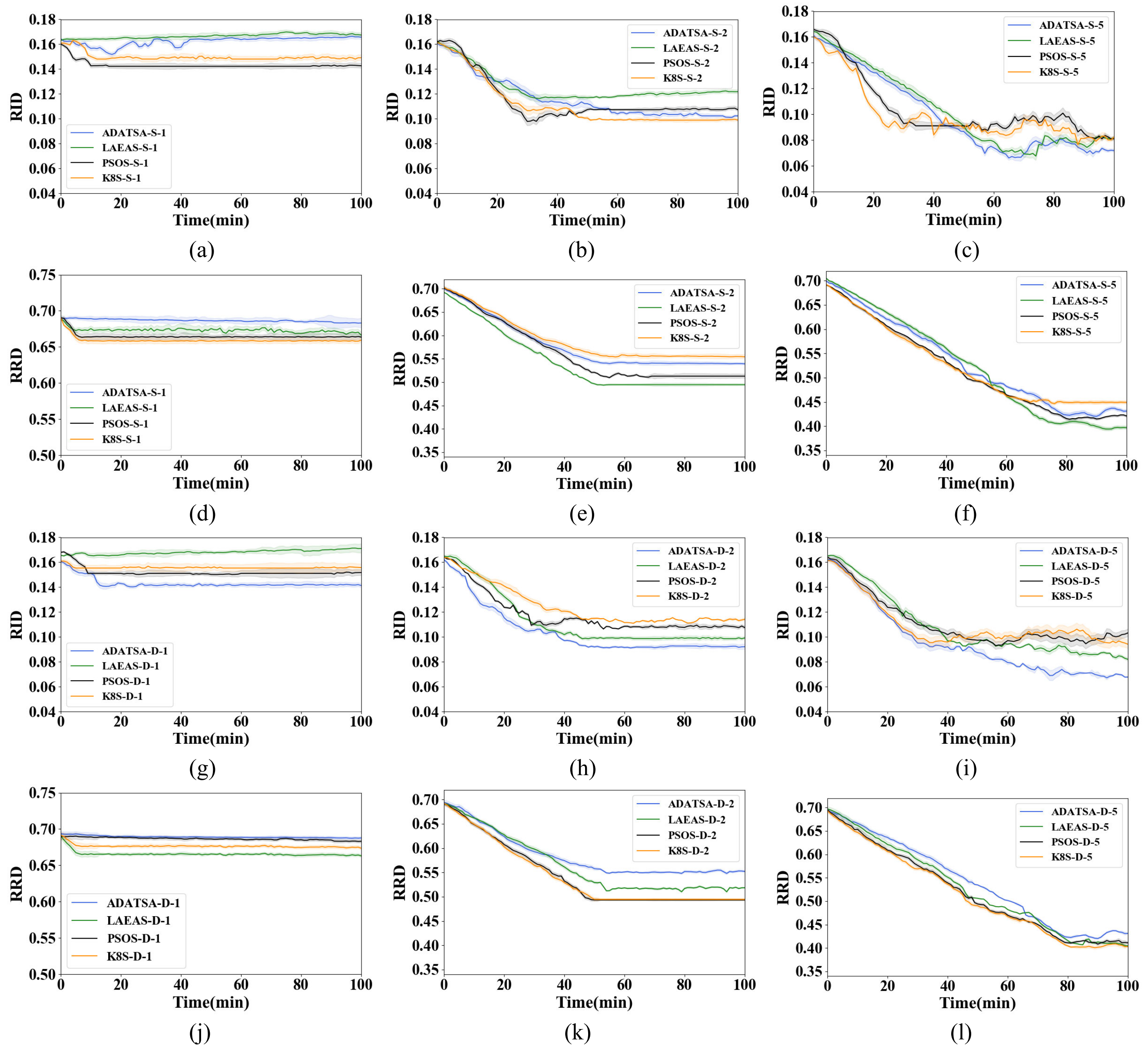


FIGURE 7. Impact of the time on convergence performance of static(-S) and dynamic(-D) scheduling under different number of tasks. Where −1, −2 and −5 represent the curves of 1×10^3, 2×10^3 and 5×10^3 tasks respectively.

after 15 minutes. In the dynamic scheduling scenario with 2×10^3 tasks, both ADATSA and LAEAS converge in 45min, but ADATSA's *RID* convergence value is even smaller. The convergence time of PSOS and K8S is about 50min. What's more, their *RID* convergence value is larger. Compared with LAEAS with the best performance and K8S with the worst performance, ADATSA achieves about 6.73% and 19.01% of *RID* performance improvement respectively. In the dynamic scheduling scenario with 5×10^3 tasks, the *RID* performance of ADATSA starts to be significantly better than all comparison algorithms after 50 minutes, and finally converges to about 0.067. In addition, we can see from Fig. 7 (j) to (l) that the *RRD* curve of ADATSA is generally above all the comparison algorithms, it shows better resource utilization efficiency. To sum up, for different number of dynamic scheduling scenarios, ADATSA has better resource utilization efficiency than LAEAS, PSOS and K8S, which can achieve balanced and convergent utilization of various resources in a short time. Thanks to ADATSA's accurate update mechanism of scheduling action, the current state of node resources and the historical migration state of tasks are effectively used to adjust the selection probability of scheduling action, which improves the convergence performance of the algorithm. Due to the defect of probability update, LAEAS algorithm needs to spend more time scheduling optimization. And PSOS searches optimal solution from the

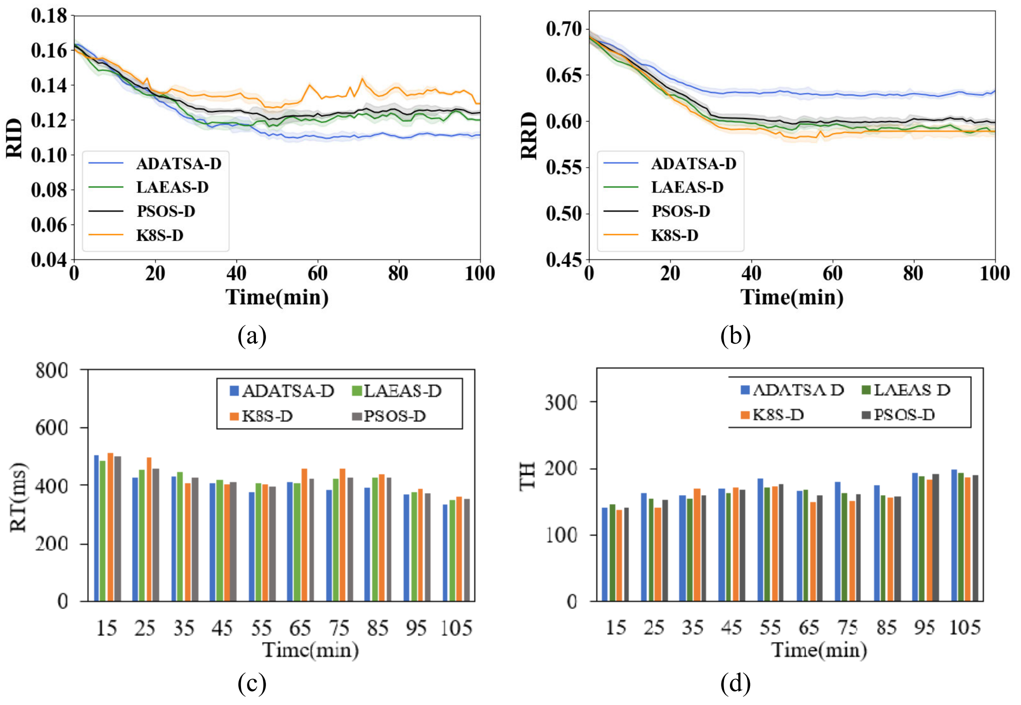


FIGURE 8. Impact of resource environment state on quality of service.

whole solution space, which not only takes much optimization time but also brings scheduling complexity.

In experiment E, we verify the impact of resource environment state on service access quality. Based on experiment D (the number of tasks is 5×10^3), a certain number of webservice applications was selected on each node for QoS testing. Specifically, 5 minutes after the end of task dynamic migration (to ensure the stable running of the application). The client uses Apache JMeter to make concurrent calls to the working nodes at regular intervals and makes QoS records. The number of concurrent requests for each application is set to 1000. The whole process is carried out by automatic script. Verify performance of ADATSA and comparison algorithms under four metrics of *RT, TH, RID* and *RRD*. The experimental results are shown in Fig. 8 (a) to (d). The experimental results show that ADATSA is better than LAEAS, PSOS and K8S on *RT, TH, RID* and *RRD* by and large. As the scheduling continues, the *RT* metric of ADATSA has a downward trend, and *TH* has been improved accordingly. However, the *RT* metric of LAEAS, PSOS and K8S fluctuated significantly, especially the *RT* metric of K8S peaked in different degrees from 65min to 85min. This is mainly due to the resource fluctuation caused by the dynamic eviction of K8S. It can be seen from Fig. 8 (a) that in the range of 60min to 80min, the resource utilization of K8S is unbalanced, which affects the application performance to a certain extent. Data analysis shows that compared with LAEAS, PSOS and K8S, the *RT* metric of ADATSA has improved by about 2.05%, 2.10% and 4.72% performance, the *TH* metric has also improved the performance by about 4.07%, 4.04% and 6.37% respectively. This is mainly because ADATSA establishes reasonable probability reward-penalty rules based on learning automata for container cloud task scheduling scenarios. Through high load task screening and secondary scheduling adjustment, ADATSA realizes the balanced utilization of cluster resources and improves the quality of service to a certain extent.

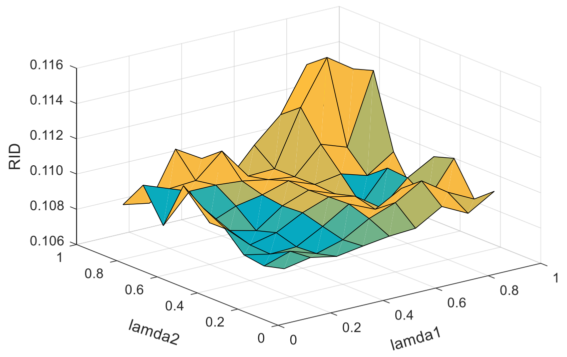


FIGURE 9. Impact of reward and penalty factors on RID.

In experiment F, we verify the impact of reward factor λ_1 and penalty factor λ_2 on the final scheduling results. Fig. 9 shows the impact of reward and penalty factors on *RID*. The three-dimensional graph is concave in front and convex in back, which indicates that too small or too large reward and penalty factors are not conducive to the final convergence of resource state. If the parameters are too small, the granularity of reward-penalty for scheduling action is not enough, and the algorithm needs a long time to converge; otherwise, the granularity of reward-penalty for scheduling action is too large, the algorithm is easy to fall into local optimum, and it is difficult to obtain the optimal scheduling. In the current experimental environment, good scheduling results can be obtained when the reward and penalty factors are set between 0.2 and 0.4.

VI. SUMMARY AND FUTURE EXPECTATION

In this paper, we present a self-adapting task scheduling algorithm (ADATSA) based on learning automata. The algorithm effectively utilizes the reinforcement learning ability of learning automata, and realizes an effective reward-penalty mechanism for scheduling actions in combination with the idle state of resources and the running state of tasks in the current environment. Meanwhile, we design a framework of task load monitoring for real-time monitoring of environment and scheduling evaluation feedback, and establish a buffer queue to achieve priority scheduling. Finally, we design comparative experiment on Kubernetes platform to simulate different scheduling scenarios, and make a comprehensive evaluation of the proposed ADATSA algorithm with learning automata based algorithm LAEAS, non-automata technology based algorithm PSOS and K8S scheduling engine in terms of resource imbalance degree, resource residual degree and QoS. Experimental results show that the proposed ADATSA algorithm has a static scheduling capability close to that of Kubernetes scheduling engine, and slightly less than heuristic scheduling optimization algorithms such as PSOS. Especially in the aspect of dynamic scheduling, compared with LAEAS, PSOS and K8S, ADATSA shows better environment adaptability, resource optimization efficiency and QoS performance.

However, there still exists some challenges that need to be overcome in the future work. For example, we use the experimental method to determine the best combination of reward-penalty factors in the container cloud environment as a reference for initialization parameters of ADATSA. However, the container cloud environment is a highly variable stochastic environment, and the environment model learned from the fixed reward-penalty factors may not be optimal. In addition, our algorithm does not consider the heterogeneity of cloud resources. Generally, users' requests for cloud resources are different, and the completion of user tasks is usually realized by multiple heterogeneous cloud resources. Therefore, the difference between heterogeneous cloud resources has a certain reference value in cloud resource optimization scheduling. In the following research, we will not only consider adding deep reinforcement learning method to further decompose and refine the model parameters to improve engineering practice ability of the algorithm, but also consider the heterogeneity of cloud resources to expand the application scenarios of the algorithm and meet different business need.

REFERENCES

[1] S. Singh and N. Singh, "Containers & docker: Emerging roles & future of cloud technology," in *Proc. 2nd Int. Conf. Appl. Theor. Comput. Commun. Technol. (iCATccT)*, 2016, pp. 804–807, doi: 10.1109/ICATCCT.2016.7912109.

[2] B. Burns, B. Grant, D. Oppenheimer, E. Brewer, and J. Wilkes, "Borg, omega, and kubernetes," *Commun. ACM*, vol. 59, no. 5, pp. 50–57, Apr. 2016.

[3] S, Pankaj, M. Govindaraju, S. Marru, and M. Pierce, "Integrating apache airavata with docker, marathon, and Mesos," *Concurrency Comput. Pract. Exper.*, vol. 28, no. 7, pp. 1952–1959, 2016.

[4] E. Casalicchio, "Autonomic orchestration of containers: Problem definition and research challenges," in *Proc. 10th EAI Int. Conf. Perform. Eval. Methodologies Tools*, 2017, pp. 1–4.

[5] M. Lin, J. Xi, W. Bai, and J. Wu, "Ant colony algorithm for multi-objective optimization of container-based microservice scheduling in cloud," *IEEE Access*, vol. 7, pp. 83088–83100, 2019.

[6] Y. Alahmad, T. Daradkeh, and A. Agarwal, "Availability-aware container scheduler for application services in cloud," in *Proc. IEEE 37th Int. Perform. Comput. Commun. Conf. (IPCCC)*, Nov. 2018, pp. 1–6, doi: 10.1109/PCCC.2018.8711295.

[7] P. F. Yang, "Research and implementation of dynamic resource scheduling based on Kubernetes," M.S. thesis, Dept. CST, Zhe Jiang Univ., Zhe Jiang, China, 2017.

[8] R. Tang, "Research on resources scheduling strategy of container cloud platform based on Kubernetes," M.S. thesis, Dept. Elect. Eng., UESTC Univ., Si Chuan, China, 2017.

[9] P. Y. Zhang and M. C. Zhou, "Dynamic cloud task scheduling based on a two-stage strategy," *IEEE Trans. Autom. Sci. Eng.*, vol. 13, no. 1, pp. 11–23, Jan. 1995.

[10] R. S. Sutton and A. G. Barto, "Barto, reinforcement learning," *A Bradford Book*, vol. 15, no. 7, pp. 665–685, 1998.

[11] K. S. Narendra and M. A. L. Thathachar, "Learning automata—A survey," *IEEE Trans. Syst. Man Cybern.*, vol. SMC-4, no. 4, pp. 323–334, Jul. 1974.

[12] T. T. Cheng, "Learning automata and its application in stochastic point location problem," M.S. thesis, Dept. Elect. Eng., SJT Univ., Shang Hai, China, 2014.

[13] K. Gong, Y. W. Wu, and K. Chen, "Container cloud multi-dimensional resource utilization balanced scheduling," *Appl. Res. Comput.*, vol. 37, no. 4, pp. 148–152, 2020.

[14] X. L. Xie and Q. Wang, "A scheduling algorithm based on multi-objective container cloud task," *J. Shandong Univ. (Eng. Sci.)*, vol. 50, no. 4, pp. 14–21, 2020.

[15] S. Pandey, L. Wu, S. M. Guru, and R. Buyya, "A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments," in *Proc. 24th IEEE Int. Conf. Adv. Inf. Netw. Appl.*, Apr. 2010, pp. 400–407, doi: 10.1109/AINA.2010.31.

[16] P. Zhou, B. Yin, X. S. Qiu, S. Y. Guo, and L. M. Meng, "Service reliability oriented cloud resource scheduling method," *Acta Electronica Sinica*, vol. 47, no. 5, pp. 1036–1043, 2019.

[17] L. Zuo, L. Shu, S. Dong, C. Zhu, and T. Hara, "A multi-objective optimization scheduling method based on the ant colony algorithm in cloud computing," *IEEE Access*, vol. 3, pp. 2687–2699, 2015.

[18] H. Li, X. Wang, S. Gao, and N. Tong, "A service performance aware scheduling approach in containerized cloud," in *Proc. IEEE 3rd Int. Conf. Comput. Commun. Eng. Technol. (CCET)*, Aug. 2020, pp. 194–198.

[19] S. Liu and N. Wang, "Collaborative optimization scheduling of cloud service resources based on improved genetic algorithm," *IEEE Access*, vol. 8, pp. 150878–150890, 2020.

[20] Q. Liu, J. Li, and H. Lv, "Edge-cloud collaborative optimization scheduling with micro-service architecture," *J. Comput. Commun.*, vol. 7, no. 10, pp. 1036–1043, 2019.

[21] C. Y. Li, Y. Song, and J. T. Ma, "RIOPSO algorithm for fuzzy cloud resource scheduling problem," *J. Frontiers Comput. Sci. Technol.*, vol. 40, no. 4, pp. 1–14, 2020.

[22] C. F. Jian, J. W. Chen, and M. Y. Zhang, "Improved chaotic bat swarm cooperative scheduling algorithm for edge computing," *J. Chin. Comput. Syst.*, vol. 40, no. 11, pp. 2424–2430, 2019.

[23] J. Chang, Z. Hu, Y. Tao, and Z. Zhou, "Task scheduling based on dynamic non-linear PSO in cloud environment," in *Proc. IEEE 9th Int. Conf. Softw. Eng. Service Sci. (ICSESS)*, Beijing, China, Nov. 2018, pp. 877–880, doi: 10.1109/ICSESS.2018.8663825.

[24] R. Xu, W. D. Wang, X. Y. Gong, and X. R. , "Delay-aware resource scheduling optimization in network function virtualization," *J. Comput. Res. Develop.*, vol. 55, no. 4, pp. 738–747, 2018.

[25] D. J. Kong, "Kubernetes resource scheduling strategy for 5G multi-access edge computing," *Comput. Eng.*, vol. 44, no. 3, pp. 89–97, Mar. 2017.

[26] X.-L. Shi and K. Xu, "Utility maximization model of virtual machine scheduling in cloud environment," *Chin. J. Comput.*, vol. 36, no. 2, pp. 252–262, Mar. 2014.

[27] Y. Shi, L. Luo, and H. Guang, "Research on scheduling of cloud manufacturing resources based on bat algorithm and cellular automata," in *Proc. IEEE Int. Conf. Smart Manuf., Ind. Logistics Eng. (SMILE)*, Apr. 2019, pp. 174–177, doi: 10.1109/SMILE45626.2019.8965317.

[28] S. P. Zhang and W. B. Zhong, "Cloud resource schedule based on cellular ant colony optimization," *Microelectron. Comput.*, vol. 32, no. 8, pp. 54–57, 2015.

[29] J. Gasior and F. Seredynski, "Dynamic job scheduling in the cloud using slowdown optimization and sandpile cellular automata model," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. Workshop*, May 2015, pp. 276–285, doi: 10.1109/IPDPSW.2015.139.

[30] S. Ghanavati, J. Abawajy, and D. Izadi, "Automata-based dynamic fault tolerant task scheduling approach in fog computing," *IEEE Trans. Emerg. Topics Comput.*, early access, Oct. 26, 2020, doi: 10.1109/TETC.2020.3033672.

[31] S. Sahoo, B. Sahoo, and A. K. Turuk, "An energy-efficient scheduling framework for cloud using learning automata," in *Proc. 9th Int. Conf. Comput., Commun. Netw. Technol. (ICCCNT)*, Jul. 2018, pp. 1–5, doi: 10.1109/ICCCNT.2018.8493692.

[32] A. Yazidi, I. Hassan, H. L. Hammer, and B. J. Oommen, "Achieving fair load balancing by invoking a learning automata-based two-time-scale separation paradigm," *IEEE Trans. Neural Netw. Learn. Syst.*, early access, Aug. 5, 2020, doi: 10.1109/TNNLS.2020.3010888.

[33] M. Jahanshahi, M. R. Meybodi, and M. Dehghan, "A new approach for task scheduling in distributed systems using learning automata," in *Proc. IEEE Int. Conf. Autom. Logistics*, Aug. 2009, pp. 62–67.

[34] X. L. Ma, "Research on resource scheduling strategy based on Kubernetes container cluster," M.S. thesis, Dept. Softw. Theory Comput., XAST Univ., Xi'an, China, 2019.

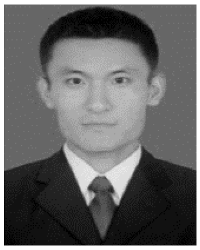
[35] H. Shen and L. Chen, "A resource usage intensity aware load balancing method for virtual machine migration in cloud datacenters," *IEEE Trans. Cloud Comput.*, vol. 8, no. 1, pp. 17–31, Jan. 2020.

[36] Z. Zheng, Y. Zhang, and M. R. Lyu, "Investigating QoS of real-world Web services," *IEEE Trans. Services Comput.*, vol. 7, no. 1, pp. 32–39, Jan. 2014.

[37] A. Gorbenko and V. Popov, "Task-resource scheduling problem," *Int. J. Autom. Comput.*, vol. 9, no. 4, pp. 429–441, Aug. 2012.

[38] Z. Yang, C. Yin, and Y. Liu, "A cost-based resource scheduling paradigm in cloud computing," in *Proc. 12th Int. Conf. Parallel Distrib. Comput., Appl. Technol.*, Oct. 2011, pp. 417–422.

[39] J. J. Lotf, S. H. Hosseini Nazhad, and M. Hosseinzadeh, "Applications of learning automata in wireless sensor networks," in *Proc. 5th Int. Conf. Appl. Inf. Commun. Technol. (AICT)*, 2011, pp. 1–5.



LILU ZHU was born in 1988. He is currently pursuing the Ph.D. degree with the School of Information Science and Technology, University of Science and Technology of China, Hefei, China. His main research interests include spatio-temporal data service and distributed system architecture.

KAI HUANG was born in 1988. He received the M.E. degree in computer science and technology from Jiangnan University, Wuxi, China. He is currently an Intern Researcher with the Institute of Electronics, Chinese Academy of Sciences, Suzhou. His main research interest includes container cloud.

YANFENG HU received the B.S. degree from Xidian University, Xi'an, China, in 1999, and the Ph.D. degree from the Institute of Electronics, Chinese Academy of Sciences, Beijing, in 2005. He is currently a Researcher with the Institute of Electronics, Chinese Academy of Sciences, Suzhou, China. His research interests include natural language processing and remote sensing image understanding.

XIANQING TAI received the Ph.D. degree from ZheJiang University in 1996. He is currently a Researcher with the Institute of Electronics, Chinese Academy of Sciences, Suzhou, China. His research interests include cloud computing and remote sensing image understanding.