frontiers | Frontiers in Genetics

TYPE Original Research
PUBLISHED 10 October 2022
DOI 10.3389/fgene.2022.964784

OPEN ACCESS

EDITED BY
Maurice H. T. Ling,
Temasek Polytechnic, Singapore

REVIEWED BY
Fan Jiang,
Xi'an University of Posts and Telecommunications, China
Guolin Sun,
University of Electronic Science and Technology of China, China

*CORRESPONDENCE
Lei Yu,
yuleiimu@sohu.com
Yucong Duan,
duanyucong@hotmail.com

SPECIALTY SECTION
This article was submitted to Computational Genomics, a section of the journal Frontiers in Genetics

RECEIVED 09 June 2022
ACCEPTED 21 September 2022
PUBLISHED 10 October 2022

CITATION
Yu L, Yu PS, Duan Y and Qiao H (2022), A resource scheduling method for reliable and trusted distributed composite services in cloud environment based on deep reinforcement learning. *Front. Genet.* 13:964784. doi: 10.3389/fgene.2022.964784

A resource scheduling method for reliable and trusted distributed composite services in cloud environment based on deep reinforcement learning

Lei Yu[1]*, Philip S. Yu[2], Yucong Duan[3]* and Hongyu Qiao[1]

[1]Department of Computer Science, Inner Mongolia University, Hohhot, China, [2]Department of Computer Science, University of Illinois at Chicago (UIC), Chicago, IL, United States, [3]College of Computer Science and Technology, Hainan University, Haikou, China

With the vigorous development of Internet technology, applications are increasingly migrating to the cloud. Cloud, a distributed network environment, has been widely extended to many fields such as digital finance, supply chain management, and biomedicine. In order to meet the needs of the rapid development of the modern biomedical industry, the biological cloud platform is an inevitable choice for the integration and analysis of medical information. It improves the work efficiency of the biological information system and also realizes reliable and credible intelligent processing of biological resources. Cloud services in bioinformatics are mainly for the processing of biological data, such as the analysis and processing of genes, the testing and detection of human tissues and organs, and the storage and transportation of vaccines. Biomedical companies form a data chain on the cloud, and they provide services and transfer data to each other to create composite services. Therefore, our motivation is to improve process efficiency of biological cloud services. Users' business requirements have become complicated and diversified, which puts forward higher requirements for service scheduling strategies in cloud computing platforms. In addition, deep reinforcement learning shows strong perception and continuous decision-making capabilities in automatic control problems, which provides a new idea and method for solving the service scheduling and resource allocation problems in the cloud computing field. Therefore, this paper designs a composite service scheduling model under the containers instance mode which hybrids reservation and on-demand. The containers in the cluster are divided into two instance modes: reservation and on-demand. A composite service is described as a three-level structure: a composite service consists of multiple services, and a service consists of multiple service instances, where the service instance is the minimum scheduling unit. In addition, an improved Deep Q-Network (DQN) algorithm is proposed and applied to the scheduling algorithm of composite services. The experimental results show that applying our improved DQN algorithm to the composite services scheduling problem in the container cloud environment can effectively reduce the completion time of the composite services. Meanwhile, the method improves Quality of Service (QoS) and resource utilization in the container cloud environment.

KEYWORDS

composite services, container cloud, deep reinforcement learning, service scheduling, artificial intelligence

1 Introduction

With the rapid development and popularity of the Internet, the number of network users is also increasing, but the resources of the data center decreased relatively. The development of cloud computing technology has led to a great convenience of information processing, and users can obtain reliable services through the cloud platform on a large number of data centers. However, as the composite service requested by the users are complex and diversified, the number of requests is increasing. Especially in the field of bioinformatics, biomedical research relies on a large amount of genomic and clinical data. Biomedical companies form a data chain on the cloud, and they provide services and transfer data to each other to form composite services. In such a dynamic environment, resource management and performance optimization have become a significant challenge for cloud and application providers, who not only consider user Quality of Service (QoS) but also consider the load balancing of the data center, resource utilization and problems such as energy consumption (Almansour and Allah, 2019). Therefore, an efficient and reasonable service scheduling method becomes essential for the cloud computing platform.

In addition, many cloud platforms currently use virtual machines as the underlying virtualization technology. Additional operating systems carried by virtual machines will bring performance losses to the cloud platform, and the startup speed of virtual machines is slow, so it is difficult for them to make rapid scaling responses to service load (Barik et al., 2016). As the virtualization technology at the operating system level, the container technology has minimal additional resource overhead, shorter startup and destruction time, and the performance of disk IO and CPU of the container is even close to that of the host (Joy, 2015). Therefore, it is considered to be a better solution for application distribution and deployment on the cloud platform (Bernstein, 2014). Most of the research on service scheduling is based on virtual machines, while the research on composite service scheduling based on container cloud environment is in the exploratory stage. Because the container has the characteristics of fast startup, strong migration ability, low-performance cost, and high resource utilization (Joy, 2015), it is of great value and significance to take the container as the virtualized computing resource of the cloud platform to solve the service scheduling problem. We need a model and an algorithm that can be applied to the container cloud environment to reduce the completion time of the composite service, satisfy the user service quality as much as possible, and improve the resource utilization target of the cloud platform.

Therefore, we proposed a novel composite service scheduling model and algorithm according to container instance mode which mixed reservation and on-demand. In addition, the DQN (Deep Q-Network) algorithm is improved by combining the three algorithms Dueling-DQN (Wang et al., 2016), Double-DQN (DDQN) (Van Hasselt et al., 2016), and Prioritized Experience Replay (PER) (Schaul et al., 2016). DDQN improved the training algorithm by decoupling action selection and value function evaluation. Although it is not entirely decoupled, it effectively reduced over-estimation and made the algorithm more robust. PER introduced a new learning mechanism to solve the sampling problem of experience replay and innovatively took Temporal Difference (TD) deviation as an essential consideration to ensure that important experience can be replayed first, and the priority experience replay was applied to DQN and DDQN. The learning efficiency is greatly improved. Dueling DQN is an improvement of the neural network structure, which can decouple the value and advantages of the DQN. Although the value function and the advantage function can no longer be perfectly represented as the value function and the advantage function in semantics, the accuracy of the strategy evaluation was improved, and it can be combined with other algorithms due to the strong versatility. Thus, the management of the DQN algorithm is improved. From the three levels of training algorithm, learning mechanism, and neural network structure, three improvements have been made based on DQN, but its implementation is more complex than these three algorithms. The improved DQN algorithm is used as the scheduling decision method under our model to reduce the completion time of the composite service and improve the user QoS and resource utilization of the cloud platform.

The contributions of this paper include: A new composite service scheduling model is built for container instance mode which mixed reservation and on-demand. The model considers many features, such as container storage, computing speed, network bandwidths and data streams of service output, etc. Furthermore, the model is suitable for Map-Reduce based services in distributed environments.

A new composite service scheduling algorithm is proposed, which can effectively reduce the completion time of the composite services. Meanwhile, the method improves Quality of Service (QoS) and resource utilization in the container cloud environment.

2 Related work

Cloud computing technology has greatly promoted the transformation of various industries and the development of

technological innovation. With the advancement of medical technology, the field of biomedicine has ushered in the era of big data (Yang et al., 2021). The application of cloud computing in biomedicine is becoming more and more perfect. Myers et al. (2020) developed an R package, LDlinkR, which leverages the computing resources of the cloud by harnessing the storage capacity and processing power of the LDlink web server to calculate computationally expensive LD statistics. Service scheduling, as an effective method to satisfy Quality of Service (QoS), which can rationally allocate resources and reduce energy consumption in cloud environment, has always been a research hotspot of scholars in various fields (Kyaw and Phyu, 2020). At the same time, scheduling in the cloud environment is a multi-constraint, multi-objective and multi-type optimization problem (Chen et al., 2019). Some traditional scheduling algorithms, such as Round-Robin (RR) scheduling algorithm and Least Connection (LC) algorithm, do not consider the actual load and connection status of the work node. Scheduling problem can be regarded as the problem of finding the optimal one or a group of computing resources in a limited set of computing resources under the condition of satisfying multiple constraint objectives. Heuristic algorithm is the most widely used method to solve such combinatorial optimization problems (Bernstein, 2014). The common ones are Ant Colony (AC) algorithm, Particle Swarm Optimization (PSO) algorithm, Genetic Algorithm (GA), etc. Therefore, many scholars are solving the problem of service scheduling in cloud platforms by optimizing and improving heuristic algorithms.

Panwar et al. (2019) combined Technique for Order Preference by Similarity to an Ideal Solution (TOPSIS) algorithm and PSO algorithm to divide task scheduling into two phases, which reduces the makespan of tasks and improves resource utilization of cloud platform. Chen et al. (2019) modeled the cloud workflow scheduling problem as a multi-objective optimization problem that takes both execution time and execution cost into account, and proposed a multi-objective ant colony system based on the co-evolutionary multi-population and multi-objective framework, in which two ant colony algorithms were adopted to deal with the two objectives, respectively. Cui and Xiaoqing (2018) proposed a workflow tasks scheduling algorithm based on a genetic algorithm. It plays an optimal role in the execution time of the optimal allocation scheme. George et al. (2016) adopted the Cuckoo Search algorithm to complete the assignment of tasks with the optimization goal of minimizing the computation time of tasks. Ghasemi et al. (2019) proposed a workflow scheduling method based on the Firefly Algorithm (FA), aiming at minimizing the processing time and transmission cost of workflow.

Compared with traditional scheduling algorithms, heuristic algorithms have a stronger ability for exploration and optimization. The above improvements of heuristic not only inherited the advantages of heuristic algorithms in solving combinatorial optimization problems but also solved some problems of heuristic algorithms themselves to some extent. However, these algorithms still have some problems, such as the weight coefficients of resources according to subjective experiences, slow convergence, and easily falling into local optimal solutions.

Considering the uncertainty of user requests, the dynamic nature of computing resources, the heterogeneity of cloud platforms, and many other factors, it has higher requirements for cloud platform service scheduling strategy. In recent years, with the development of artificial intelligence-related technologies, Deep Reinforcement Learning (DRL) has shown strong perception and continuous decision-making ability when dealing with automatic control problems (Orhean et al., 2018), and many scholars have begun to apply it to resource allocation and service scheduling strategies in cloud environments. Li and Hu (2019) described job scheduling as a packing problem, used DRL algorithm to calculate the fitness of jobs and machine nodes, and selected reasonable machines for jobs according to the fitness. Finally, through experiments, it proved the superiority of deep reinforcement learning as a scheduling algorithm. Cheng et al. (2018) designed a two-level scheduler combining resource allocation and task scheduling based on Deep Q-Learning, which greatly reduced the energy consumption of the cloud platform while maintaining a low task rejection rate. Wei et al. (2018) proposed an intelligent QoS aware job scheduling framework based on Deep Q-Learning algorithm, which can effectively reduce the average response time of jobs under varying loads and improve user satisfaction. Meng et al. (2019) designed an adaptive online scheduling algorithm by combining reinforcement learning with DNN, which significantly improved the scheduling efficiency of server-side task queues. Ran et al. (2019) used the Deep Determining Policy Gradient (DDPG) algorithm to find the optimal task assignment scheme meeting the requirements of the Service Level Agreement (SLA). Zhang et al. (2019) proposed a parallel execution multi-task scheduling algorithm based on deep reinforcement learning. And compared with least connection and particle swarm optimization, this algorithm significantly reduces the completion time of the job. Dong et al. (2020) proposed a task scheduling algorithm based on DRL, which can dynamically schedule tasks that have priority relationships in the cloud server, thus minimizing the task execution time and effectively solving the task scheduling problem in the cloud manufacturing environment.

Based on the above work, both the heuristic algorithm and deep reinforcement learning algorithm show their respective advantages in solving scheduling problems in cloud environments. However, there are still some problems that

TABLE 1 Summary of reviewed papers related to the task scheduling in the cloud computing.

Algorithm	Core issues to be solved	Algorithm idea	Advantage
Dueling-DQN Wang et al. (2016)	Solved the problem that in some states, action is of low importance to the overall result, and distinguished the change of Q value caused by action and state	Improved the architecture, the idea of advantage was added to evaluate the advantage function	Ensured that the relative ranking of the dominant functions of each action in this state remains unchanged
		Analyzed the advantages and disadvantages of state and action, respectively	Narrowed the range of Q value. Removed excess degrees of freedom. Improved the stability of the algorithm
DDQN Van Hasselt et al. (2016)	Solved the problem of overestimation in DQN algorithm	The idea of Double Q-learning is to reduce overestimations by decomposing the max operation in the target into action selection and action evaluation	More stable training results
			Reduced the error caused by variance
PER Schaul et al. (2016)	Changed the selection method of samples in experience replay	Improved the experience buffer training strategy	More robust
	Solved the problem of local optimization	More robust	Improved the performance of DDQN
	Offset the impact of sample distribution	Added weight to the original gradient update in SGD	Simple implementation
MOACS Chen et al. (2019)	Optimized execution time and cost	Two ant colonies are adopted to optimize execution time and execution cost, respectively	MOACS has better global search ability, particularly when dealing with large-scale workflows
		A new pheromone update rule is designed. The CHS is proposed to ensure the quality of the other objective	MOACS can generate a solution with similar WET but lower WEC than the other approaches
TOPSIS–PSO Panwar et al. (2019)	Improved the execution time, maximum completion time, resource utilization, processing cost, and transmission time in the process of task scheduling	The task scheduling is performed in two phases	Improved average resource utilization
		TOPSIS method calculates the RC of VMs with respect to each task	Low processing cost
		The PSO algorithm receives the calculated RC of each task which acts as FV of tasks (particles)	Reduced makespan for tasks
Workflow tasks scheduling optimization based on genetic algorithm Cui and Xiaoqing (2018)	Applicable to cloud computing environment combining task characteristics and resource characteristics	Assigned priority to each task	Reduced workflow scheduling cost
		Workflow tasks were divided into different levels, and a two-dimensional coding method was designed	
	Reduced the execution cost of workflow task scheduling	A new genetic crossover and mutation operation were designed to produce new different offspring, so as to increase population diversity	
FA Ghasemi et al. (2019)	Optimized the cost of executing the whole workflow and load balancing among workstations	The position of each firefly represents the feasible solution to a problem to be solved, and the brightness of the firefly represents the fitness of the firefly's position	Minimized the processing time
		Each firefly flies towards a firefly that looks brighter than itself	Reduced transmission cost of workflow
An intelligent QoS-Aware Job Wei et al. (2018)	Met the QoS requirements of users	Learnt to make appropriate online job-to-VM decisions for continuous job requests directly from its experiences without any prior knowledge	Reduced the average response time of jobs under different loads. Improved user satisfaction

(Continued on following page)

TABLE 1 (*Continued*) Summary of reviewed papers related to the task scheduling in the cloud computing.

Algorithm	Core issues to be solved	Algorithm idea	Advantage
Scheduling and resource management algorithm for multi-user mobile-edge computing systems Meng et al. (2019)	The problem of delay-sensitive task scheduling and resource (e.g., CPU, memory) management on the server side in multi-user MEC scenario	Built a system that learns to manage resources directly from experience by using reinforcement learning with adaptive policy iteration represented *via* DNN. Designed a new reward function to reduce average slowdown and average timeout period of tasks in the queue	Reduced average slowdown and average timeout period of tasks in the queue Improved the scheduling efficiency of server-side task queue
DDPG Ran et al. (2019)	Model free strategy for learning continuous action	DDPG combines the ideas of DPG and DQN It used the experience replay and delayed update target network in DQN It can run in continuous action space based on DPG	DDPG can run in a continuous action space Solved the classical inverted pendulum control problem Met service level agreements
MDTS Zhang et al. (2019)	The problem of scheduling jobs with scalable parallel tasks in general parallel computing systems, where there is a demand to determine the task placement plan with the goal of minimizing the job completion time, the load imbalance value, and the total cost	Within each task-specific branch, there is a fully connected layer and an output layer	Reduced the job completion time and optimized the load balancing problem. Improved task scheduling performance. MDTS is superior to the raw DRL algorithm
Data-dependent tasks re-scheduling energy efficient algorithm Xiaoqing et al. (2018)	Reduced energy consumption in the data center	Set the task priority to the sum of the upper and lower values of the task Used the task priority to calculate the critical path and critical resources of the task graph Calculated the energy efficiency of each resource under the initial scheduling scheme	Reduced energy consumption in the data center
DRL-based algorithms Islam et al. (2021)	Satisfied generalization to optimize multiple objectives while capturing or learning the underlying resource or workload characteristics	Two DRL-based agents (DQN and REINFORCE) DQN: An ϵ-greedy policy was used that selects the greedy action with probability $1-\epsilon$ and a random action with probability ϵ REINFORCE: It worked by utilizing Monte Carlo roll-outs. After the collection step, the algorithm updates the underlying network using the updated policy gradient Trained them as scheduling agents in the TF-agent framework	Reduced both the total cluster VM usage cost and the average job duration
Sharer Liang et al. (2020)	Improved the efficiency of resource management in CMfg	The proposed model transformed metrics generated from the individual needs of multiple users into a multiobjective reward Proposed a blacklist mechanism and a narrow baseline to improve the learning performance of RL	Adapt to different conditions Converged quickly

have not been considered in some references when solving scheduling problems in cloud platforms. References (Almezeini and Hafez, 2017; Li and Hu, 2019; Panwar et al., 2019) only discussed a single service type without discussing the diversity of services and the correlation between services. References (Cui and Xiaoqing, 2018; Xiaoqing et al., 2018) gave the corresponding weight coefficients of each resource through subjective experience.

References (Zhang et al., 2019; Dong et al., 2020) did not take into account the transmission cost between resource nodes of the execution results of services in the actual scheduling process of composite services. In the actual environment, the data transmission time between sub-services affects the completion time and operation cost of composite services to some extent. With the increasing complexity of user requests and the increasing granularity of services, each service can be scheduled for parallel execution in multiple servers to reduce the response time of services and improve the quality of services for users. References (Orhean et al., 2018; Xiaoqing et al., 2018; Chen et al., 2019; Dong et al., 2021) did not consider the parallelism of services when discussing the problem of service scheduling. We compared some algorithms in Table 1.

In addition, most of the above studies took virtual machines as virtualized computing resources to study the problem of service scheduling, while containers have the advantages of simple deployment and fast startup speed, so it is of certain research significance and value to discuss the problem of service scheduling based on the container cloud environment.

3 System model

3.1 Problem description

Based on the container cloud environment, this section focuses on the scheduling method of composite services. In the initialization stage, a certain number of host nodes are set, and each host node initializes: 1) a certain number of reserved container instances with different configurations; 2) a certain number of on-demand containers. In the reserved mode, the container instance is in the startup state and uses the allocated resources for the scheduled services at any time. The container in on-demand mode is dormant initially and takes a period of time to be started. The composite service is defined as the three-level structure of "composite service, sub-service, instance." As the basic scheduling unit, the sub-service instance is scheduled to be executed in the container, which in essence represents the number of parallel execution of sub-services. In addition, the scheduling of sub-service instances and the starting of containers in on-demand mode are determined by the service scheduling algorithm.

3.2 Problem constraints

A composite service consists of multiple sub-services (hereinafter referred to as "Services") that have an association relationship, including the order of prior execution and data dependencies among the services. In addition, each service includes one or more service instances, and each service instance of the same service has the same physical performance requirements. A composite service can be represented by a directed acyclic graph, i.e., $CS = (SVC, E)$, where the finite set $SVC = \{svc_1, \ldots, svc_m\}$ indicates that a composite service contains $m(m \geq 1, m \in N^+)$ services. Each service has $n(n \geq 1, n \in N^+)$ service instances, denoted as $svc_i = \{st_i^1, \ldots, st_i^n\}(i \in m)$. The set of directed edges $E = \{(svc_i, svc_j)|1 \leq i, j \leq m, i, j \in N^+\}$ describes the relationship between services, (svc_i, svc_j) means that svc_i is the predecessor service of svc_j, and svc_j is called the successor service of svc_i. Only after all service instances of all precursor services of svc_j have been executed, svc_j is allowed to be scheduled and executed. The service without the precursor service is called the start service svc_{start}, and each composite services has at least one start service. Service without successor services is called end service svc_{end}. Similarly, each composite service has at least one end service. Each Roman character (e.g., I, II) represents the number of service instances contained in the corresponding service. This scenario is prevalent for Map-Reduce algorithms in distributed environments.

Each service instance will be scheduled to a container, and each service contains multiple service instances, which means that each service can be executed by multiple containers together. The characteristic definition of service svc_i can be denoted by Eq. 1, where cpu_i, mem_i, $disk_i$ represent the physical performance requirements of service svc_i, such as CPU, memory, and disk storage, respectively. $length_i$ denotes the length of the result data after the completion of the service execution; $inst_num_i$ denotes the number of service instances of service svc_i; $duration_i$ represents the expected execution time of the subservice svc_i.

$$svc_i = \{cpu_i, mem_i, disk_i, length_i, inst\_num_i, duration_i\}(i \in n) \tag{1}$$

As the smallest scheduling unit in a composite service, the service instances have the same physical resource requirements as the service it belongs to. All instances of the same service can be executed in parallel, and instances of each service are able to execute different binary files for Map-Reduce scenarios. Eq. 2 defines the *kth* service instance of svc_i.

$$st_i^k = \{k, cpu_i, mem_i, disk_i, length_i, duration_i\}(i \in n, k \in m) \tag{2}$$

3.3 Resource model

In the cloud platform, physical hosts are the infrastructure that truly provides physical resources such as CPU and memory for containers and services. All hosts in a host cluster are denoted

as $H = h_1, \ldots, h_P$, where p is the number of hosts in the cluster. $h_x(x \in p)$ represents the xth host in the host cluster, and the definition of h_x is shown in Eq. 3.

$$h_x = \{hid, cpu\_cap_x, mem\_cap_x, disk\_cap_x, bw\_cap_x, container\_num_x, cpu_x, mem_x, disk_x, bw_x\}(x \in p) \quad (3)$$

where hid represents the unique ID of the host. And cpu_cap_x, mem_cap_x, $disk\_cap_x$, bw_cap_x, respectively represent the CPU capacity, memory capacity, disk storage capacity, and bandwidth capacity of the host. $container\_num_x$ represents the maximum number of containers that can be allocated by the host h_x. cpu_x, mem_x, $disk_x$, bw_x respectively represent the remaining amount of the host's CPU, memory, disk storage, and bandwidth.

In addition, all containers in the cluster can be represented by the set $C = \{c_1, \ldots, c_q\}$, where q is the number of containers. $c_y(y \in q)$ represents the physical performance state of the yth container, and the definition of c_y is shown in Eq. 4.

$$c_y = \{cid_y, hid_y, cpu\_cap_y, mem\_cap_y, disk\_cap_y, bw_y, cpu_y, mem_y, disk_y, act_y, act\_time_y\}(y \in q) \quad (4)$$

where cid_y represents the container ID, which is the unique identifier of the container. hid_y represents the host ID to which the container c_y belongs. cpu_cap_y, mem_cap_y, $disk\_cap_y$, bw_y respectively represent the CPU capacity, memory capacity, disk capacity, and bandwidth capacity of the container c_y. cpu_y, mem_y, $disk_y$, respectively represent the remaining amount of the container's CPU, memory, and disk during operation. act_y is the judgment flag, which indicates whether the container c_y is already in the state of the host. If $act_y = 1$, means that the container c_y is in the running state, and $act_y = 0$ means that the container c_y is in the dormant state. act_time_y represents the startup time of the container.

In order to compare and analyze resource utilization from three dimensions of CPU, memory, and disk, UST_i^k is defined as the resource utilization after each service instance is scheduled. The definition of average resource utilization AVUST is shown in Eq. 5.

$$AVUST = \frac{\sum_{i=1}^{m}\sum_{k=1}^{n} UST_i^k}{number\ of\ service\ instances} \quad (5)$$

3.4 Scheduling model

Before all composite services are scheduled, the hosts and containers in the data center need to be initialized. In the initialization phase, a series of physical hosts with different configurations are first created, and each host is allocated with $container\_num_x$ containers, including different configurations of reserved and on-demand containers. The containers in the reservation mode can run the scheduled service instances at any time based on the allocated resources. The containers in the on-demand mode are in the dormant state by default, which occupies a certain amount of physical resources, but there are no remaining amount of resources. The resource state of the containers in the on-demand mode is shown in Eq. 6.

$$\begin{cases} cpu\_cap_y > 0 \\ mem\_cap_y > 0 \\ disk\_cap_y > 0 \\ bw_y > 0 \\ cpu_y = 0 \\ mem_y = 0 \\ disk_y = 0 \end{cases} \quad (6)$$

Constraints must be satisfied to schedule the service to the container for execution. When the service instance st_i^k is scheduled to the container c_y, the physical resource requirements of the service instance st_i^k must not be greater than the corresponding physical resource capacity of the container c_y, otherwise it will wait for the right resources to execute. Therefore, the constraint condition that needs to be met to dispatch the service instance st_i^k to the container c_y is shown in Eq. 7.

$$\begin{cases} cpu_k \le cpu\_cap_y \\ mem_k \le mem\_cap_y \\ disk_k \le disk\_cap_y \end{cases} \quad (7)$$

When a service svc_i is ready, all service instances of the service can be scheduled to the containers for execution one by one within the same scheduling time window. However, the resource status of the container changes from time to time as the service scheduling progresses. When the service instance is scheduled to the appropriate container, it will not be executed immediately. Because the following three steps are required:

(1) First, the status of the selected container needs to be determined. If the container has already been started, that is, $act_y = 1$, then ignore this step. Otherwise, $act_y = 0$, start the container, which will consume the time of act_time_y.
(2) After the completion of step one, it is necessary to wait for the execution result of the precursor service to be transmitted to the container. The data transmission time is related to the result data length after the execution of the precursor subservice, the bandwidth of the container, and the host. Since the precursor service has multiple service instances, each service instance will be scheduled to run in a container. It can be understood that each service can be scheduled to run in multiple containers, so it is necessary to calculate the minimum transmission time of the result data from the container scheduled by the precursor service to the container where the current service instance is located. The data transmission time between containers in the same host is negligible. The data transmission time between different hosts is directly related to factors such as container bandwidth and data length. The data transmission time is shown in Eq. 8.

$$transTi^k(c_u, c_v) = \begin{cases} 0, & u = v \ or \ hid_u = hid_v \\ ratio, & other \end{cases}$$
$$ratio = \frac{length_i}{min\ (bandwidth_u,\ bandwidth_v)} \tag{8}$$

(3) In addition to the data transmission time, it is necessary to wait for the remaining amount of the physical resources of the container to meet the physical resource requirements of the service instance itself. Record the waiting resource time of the service instance st_i^k in the container c_y as wr_i^k.

Based on the above three steps, it can be concluded that after the service instance st_i^k is scheduled, the period before execution is the total waiting time of the service instance TW_i^k:

$$TW_i^k = \begin{cases} transTi^k + wr_i^k, & act_y = 1 \\ act\_time_y + transTi^k + wr_i^k, & act_y = 0 \end{cases} \tag{9}$$

As mentioned above, the execution of the service is finished when all the instances of the service svc_i are executed. Therefore, the response time T_i of the service svc_i should be denoted as:

$$T_i = \max_k \left(T_i^k\right) \tag{10}$$

Taking the submission time of the composite services as the earliest start execution time T_{start} and the completion time T_{end} of the last service instance in the sub-service as the completion time of the composite service, thus the actual completion time TC of the entire composite service is denoted by Eq. 11.

$$TC = T_{end} - T_{start} \tag{11}$$

In order to denote the expected completion time of the composite services more conveniently, the composite service is divided into layers according to the execution order of the service. The start sub-service is placed in the first layer, and the end sub-service is placed in the last layer.

The service completion time of each level is the response time of the service with the longest response time in the level, as shown in Eq. 12, where l represents the level and u represents the number of services contained in the level.

$$TL_l = \max_u (T_i) \tag{12}$$

Define the maximum expected completion time for an entire composite service as:

$$TE = 2 \sum_v TL_l \tag{13}$$

The interaction between the user and the cloud platform takes the whole composite service as the unit, and the user can set the desired QoS demand when sending the request. The completion time of the composite service is an important QoS indicator for users, so this paper takes the maximum expected completion time of the composite services TE as the user's QoS demand. Eq. 14 indicates whether the user's demand QoS can be met:

$$success(CS) = \begin{cases} 1, & TC \le TE \\ 0, & else \end{cases} \tag{14}$$

For cloud and service providers, the goal of service scheduling is to meet users' QoS requirements as far as possible while completing service execution under the constraints of limited IaaS or PaaS resources, which needs to be implemented through an efficient online service scheduling algorithm.

4 Algorithm design and implementaion

4.1 Prioritized 3-deep Q-network

In the process of using DQN (Deep Q-Network), there will be a problem of overestimate (Liang et al., 2020). Therefore, in recent years, many scholars have proposed improved algorithms for DQN, including DDQN, Dueling DQN, distributed DQN, PER, etc. This section combines DDQN, Dueling DQN, and Prioritized Experience Replay three algorithms to improve DQN at the same time to construct Prioritized Dueling-DDQN (hereinafter referred to as Prioritized 3-DQN) algorithm. This algorithm avoids overestimation of DQN to a certain extent. At the same time, when updating the parameters of neural network, PER algorithm is used to replace the random sampling method in DQN and select the most effective learning samples from the sample memory to achieve the purpose of efficient learning.

The Prioritized 3-DQN algorithm also uses two neural networks with the same structure: the Eval network and the Target network. The Eval network is used to calculate the estimated Q value and can be updated in real time. The Target network is used to calculate the target Q value, and it is a temporarily frozen network. This article has made three improvements to DQN: two decoupling actions and one sampling method improvement. The specific descriptions are as follows:

(1) The output layer of the neural network is decoupled into two output streams, which output the current state value V and the action advantage function A, respectively, and then combine the state value V and the advantage function A to form the Q value. The advantage function refers to the degree of merit of the value that can be obtained by taking an action relative to the average value of the state for a particular state. In order to calculate the advantage function value corresponding to each action more conveniently, the average value of the advantage function value of all actions is set to 0. If the advantage function value corresponding to a certain action is greater than the average value in the state, then the advantage function

value corresponding to the action is positive, and vice versa. At this time, the calculation method of the Q value is shown in Eq. 15, where θ represents the neural network parameter, α and β represent the output flow neural network parameters corresponding to the state value and the action advantage function, and n is the action dimension.

$$Q(s,a;\theta) = V(s;\alpha) + A(s,\ a;\beta) - \frac{\sum_{a'}^{n} A(s,\ a';\beta)}{n} \tag{15}$$

(2) Based on DQN, the overestimation problem is solved by decoupling the selection of target action and calculating the target Q value. When calculating the actual value of Q, the Eval network provides the action in the next environment state, and the Target network provides the Q value of this action.

The Q value. At this time, the update process of the neural network is shown in Eq. 16, where θ and θ^{-} represent the Eval network and the Target network, respectively.

$$\begin{aligned} Q(s_t,a_t;\theta) &\leftarrow Q(s_t,a_t;\theta) \\ &+ \propto \left[r_t + \gamma Q\left(s', a^{max}\left(s';\theta\right);\theta^{-}\right) - Q(s_t,a_t;\theta)\right] \end{aligned} \tag{16}$$

(3) In the offline training phase of traditional DQN, the training samples are randomly selected from the experience replay pool without considering the priority relationship of the samples. However, different samples have different values, and the samples directly affect the training effect of the neural network. In order to improve the training effect of the neural network, it is necessary to determine a priority for each sample and conduct sampling according to the priority of the sample. As mentioned above, the Target network does not have the function of real-time updates. Therefore, as the Eval network is continuously updated, there will be a certain gap between the two networks while calculating the Q value. This gap is named the timing difference TD_Error. TD_Error can be represented by Eq. 17. The larger the TD_Error, the larger the gap between the current Q function and the target Q function, the more the neural network needs to be updated at this time, so TD_Error can be used to measure the value of the sample. In order to prevent the network from overfitting, samples can be drawn by probability. At this time, the probability of samples being drawn is shown in Eq. 18, where ϵ is a small value close to 0, which guarantees Samples with TD_Error of 0 may also have a chance to be drawn.

$$TD\_Error = r_t + \gamma Q\left(s',\ a^{max}\left(s';\theta\right);\ \theta^{-}\right) - Q(s_t,\ a_t;\ \theta) \tag{17}$$

$$P(i) = \frac{p_i}{\sum p_i} \tag{18}$$

where, $p_i = |TD_Error + \epsilon|$. The process of our Prioritized 3-DQN algorithm is as follows:

Initialize θ - network parameters; θ^{-} - copy of θ;Initiative N - the capacity of replay memory D;N_b - training batch size; C – target network replacement freq
Set $Q(s,a;\pi) = V(s;\pi,\beta) + (A(s,a;\pi,\alpha) - mean\sum_{a'} A(s,a';\pi,\alpha))$, while $\pi\epsilon(\theta,\theta^{-})$, and α, β are the parameters of the two streams of fully-connected layers
for $episode = 1$ to M **do**
  Initialize frame sequence $x \leftarrow ()$
  for $t = 1, T$ **do**
    Set state $s_t \leftarrow x$
    With probability ε select a random action a_t
    Otherwise select $a_t = argmax_a Q(s,\ a;\theta)$
    Execute action a_t in emulator and observe reward r_t and $s_{t+1} \leftarrow x$
    Store transition $(s_t,\ a_t,\ r_t,\ s_{t+1})$ in D
    Simpling N_b samples $(s_j,\ a_j,\ r_j,\ s_{j+1}),\ j\epsilon(1,\ N_b)$ from D with probability $P(j) = p_j/\sum_i (p_i)$
    Construct target values, one for each of $N_b$tuples :
    Define $a^{max}(s';\theta) = argmax_{a'} Q(s',a';\theta)$
    Set $y_j = \begin{cases} r_j & ,if\ s'isterminal \\ r_j + \gamma Q(s',\ a^{max}(s';\theta);\theta^{-}) & ,otherwise \end{cases}$
    Perform a gradient descent step on $(y_j - Q(s_j,a_j;\theta))^2$ with repect to the network parameters θ
    Update the sampling weight of each sample according to the formula $(y_j - Q(s_j,\ a_j;\theta))$
    Replace target parameters $\theta^{-} \leftarrow \theta$ every C steps
  end for
end for

Algorithm 1. Prioritized 3-DQN.

4.2 State space

When the service $svc_i$is ready, the method selects an instance of svc_i each time st_i^k and schedules it to a certain container. The environment status at this time is mainly determined by the physical relevant factors of the service instance st_i^k, such as resource requirements, running status of the container cluster are determined. Therefore, the state space can be denoted by Eq. 19:

$$S_i^k = \left[st_i^k, c_1, obs_{c_1}, pre_{c_1}^{svc_i}, \ldots, c_q, obs_{c_q}, pre_{c_q}^{svc_i}\right] \tag{19}$$

where

$obs_{c_y} = [que\_len_y, cpu\_len_y, mem\_len_y, disk\_len_y],\ (y \in q)$

Each value in the state space affects the scheduling decision of DRL, where st_i^k represents the current service instance to be scheduled, which is represented by the aforementioned Eq. 2, and cy represents the resource state of the yth container in the cluster, as shown in Eq. 4. It should be noted that there is a one-to-many relationship between service instances and containers. Each service instance can only be completed by one container, but each container can be assigned multiple service instances. When the remaining physical resources of the container are insufficient and the resource requirements of the service instance are required, the newly scheduled service instance needs to be added to the services queue to be executed in the container. obs_{c_y} is the running status of container c_y, where que_len_y represents the length of the service instance queue to be executed in container c_y, and cpu_len_y, mem_len_y, and $disk\_len_y$ respectively represent the sum of the CPU, memory, and disk storage space requirements of the waiting queue. The characteristic value $pre_{c_1}^{svc_i}$ represents the proportion of the result data length of the predecessor service of the current service instance in the container c_y after execution. For example svc_3 has two predecessor services svc_1 and svc_2. Assume that the length of the result data after the execution of these two precursor services is 4 and 6, so only the service instance of svc_1 is scheduled to the container c_1. The service instance of svc_3 is st_3^1. When being scheduled, $pre_{c_1} = 4/(4+6) = 0.4$.

TABLE 2 Table of data relation comparison.

Fields of batch_task table	Attributes of class service	Description
task_name	service_name	Service name
inst_num	inst_num	The number of instances
job_name	cs_name	The name of composite service
Duration	Duration	Expected execution time
plan_cpu	cpu	CPU cores requirements
plan_mem	mem	Memory requirements
Disk	Disk	Disk storage requirements
Length	Length	The length of result

TABLE 3 Table of dataset settings.

Dataset name	The number of composite services	The number of services	The number of service instances
Training data set	1,036	5,832	38,586
Test data set1	345	1,500	12,320
Test data set2	426	2,200	18,020
Test data set3	512	2,780	25,200

4.3 Action space

During scheduling decision-making, a suitable container is selected for the service instance as the action in DRL, and the action space is all the containers that can be selected. Suppose that the data center contains p hosts $\{h_1, \ldots, h_p\}$ at a certain time, host h_x can assign at most $container\_num_x$ containers with different configurations. When service instance st_i^k is ready to be scheduled, the agent in DRL can schedule it to any container in the cluster for execution, including all containers in reserved and on-demand modes. The action space at this time is shown in Eq. 20.

$$a_{num} = h_x \times container\_num_x \quad (x \in p) \tag{20}$$

4.4 Reward function

In order to enable the agent in DRL to learn effectively and obtain an effective scheduling strategy that optimizes the goal, a reasonable reward function needs to be designed to guide the learning process of the agent. In our model, in order to minimize the completion time and improve the user QoS and resource utilization of the cloud platform, this paper uses the difference between the expected execution time of the service instance and the waiting time. It then uses the ratio of the expected execution time as the reward for each scheduling. The value is as follows:

TABLE 4 Resource node settings.

Hosts	Containers	Detailed description (CPU cores; Memory capacity; Disk capacity; Bandwidth; Status)
Host 0	Container 0	4; 1.56; 10; 5; Running
	Container 1	4; 1.56; 10; 5; Stopped
	Container 2	8; 3.13; 18; 8; Running
Host 1	Container 3	4; 1.56; 10; 5; Stopped
	Container 4	8; 3.13; 18; 3; Running
	Container 5	8; 3.13; 18; 3; Stopped
Host 2	Container 6	4; 2.34; 12; 5; Stopped
	Container 7	8; 3.13; 18; 3; Running
Host 3	Container 8	4; 2.34; 12; 3; Running
	Container 9	8; 3.13; 18; 5; Stopped

$$r_i^k = \frac{\left(durtation_i - TW_i^k\right)}{duration_i} = 1 - \frac{TW_i^k}{duration_i} \tag{21}$$

Based on Eq. 21, the interval of reward value can be deduced as$[-\infty, 1]$. When the overall waiting time of the service instance

TABLE 5 Algorithm parameter setting.

Parameter name	Value
The number of hidden layers	3
Activation function	ReLU
Greed index ε	0.9
Experience replay pool size N	3,000
Number of sample sets N_b	200
Learning rate α	0.001
Discount factor γ	0.9
ϵ	0.001
Target network update frequency C	30

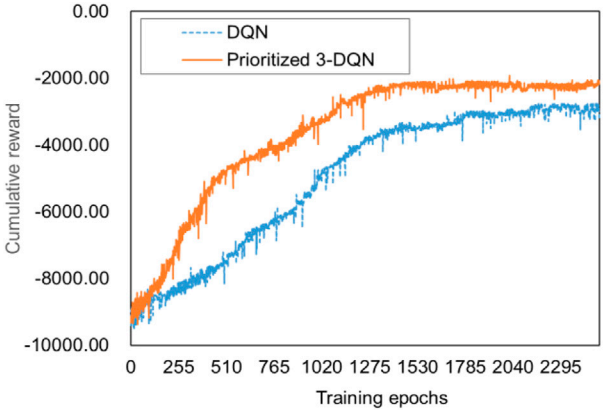

FIGURE 1
Training effect comparison chart.

TW_i^k is 0, the scheduling reward reaches the highest value of 1; when the overall waiting time TW_i^k is equal to the expected execution time, the reward value is 0; when the overall waiting time TW_i^k is greater than the expected execution time, the reward value begins to show a negative value. The longer the waiting time for execution, the smaller the reward value, and the greater the punishment. Through the reasonable design of the reward function, DRL can learn an effective service scheduling policy.

5 Experimental results

5.1 Simulation experiment setup

This paper uses Alibaba Cluster Data V2018 (Alibaba, 2018) as the data set for the simulation experiment. The data set contains six files in CSV format, describing the status information of the physical machine cluster, container cluster, and batch processing tasks. The original data set has a huge amount of data. There is inevitably a problem of missing data, and the data set is scattered and difficult to operate. Therefore, it is necessary to preprocess the original data set to obtain more targeted and valuable data. During the experiment, the preprocessed batch job data needs to be parsed and mapped into a composite service entity. The comparison between the fields of the preprocessed batch_task table and the attributes of the service class is shown in Table 2.

This paper divides the experimental data set into two parts: the training data set and the test data set, as shown in Table 3. In this experiment, 5,832 pieces of data are selected as services from the batch_task table, forming a total of 1,036 composite services, including 38,586 service instances. At the same time, to fully verify the effectiveness of Prioritized 3-DQN as a scheduling algorithm, this paper sets up three test sets with different data volumes.

In the initial stage of the simulation experiment, four hosts with different configurations are set, and each host contains container instances with different configurations and states. The relevant configuration of each container is shown in Table 4.

In implementing the Prioritized 3-DQN algorithm, the parameter settings are shown in Table 5. Both the Eval network and the Target network contain three fully connected neural network hidden layers, the last layer of which is divided into two output channels: state value and action advantage function. The greedy coefficient ε is 0.9. Each time the neural network parameters are updated, it will increase by 0.0001. That is, when selecting the container for the service instance, the container with the largest Q value will be selected with a probability of 0.9, and the container will be randomly explored with a probability of 0.1. After 1,000 updates, the value of ε becomes 1, and random exploration is no longer performed when selecting a container, but only the container corresponding to the largest Q value is selected. ϵ is set to 0.001, which ensures that samples whose timing difference TD_Error is 0 will also have a chance to be sampled. The target network update frequency C is set to 30, which means that every 30 times the Eval network is updated, its network parameters are copied to the Target network.

5.2 Prioritized 3-deep Q-network training effect

The essence of deep reinforcement learning algorithm learning is to maximize the cumulative reward of the round as the optimization goal, so the training effect can be reflected by the trend of the cumulative reward as the value changes with the number of training rounds. In addition, the Prioritized 3-DQN scheduling algorithm proposed in this

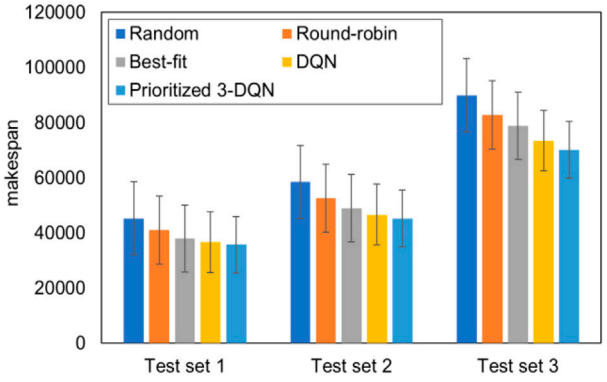

FIGURE 2
Comparison chart of average completion time of each test set with standard error.

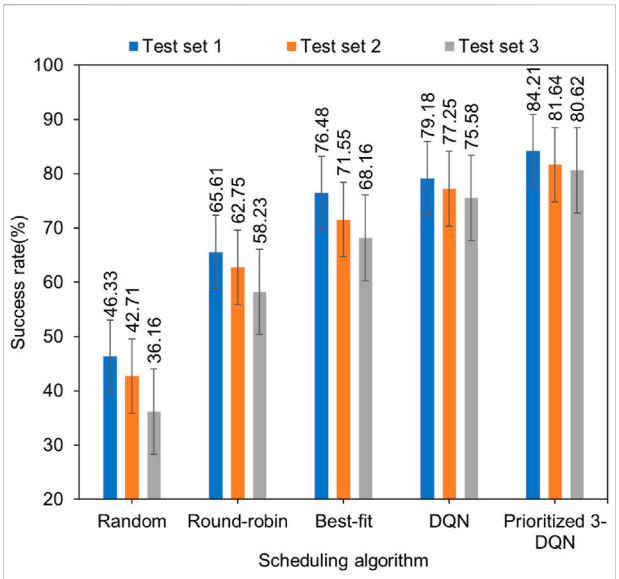

FIGURE 3
Comparison chart of composite services success rate with standard error.

paper is improved based on the DQN algorithm. In order to evaluate the convergence and stability of the improved Prioritized 3-DQN scheduling algorithm, it is compared with the original DQN algorithm. After 2,500 rounds of training using the training data set, they finally reached their optimal training effects. Figure 1 is a comparison chart of training effects.

It can be seen from Figure 1 that as the number of training rounds increases, the cumulative reward values calculated by the two algorithms show a gradual upward trend. After a certain number of rounds, they have reached a stable trend, indicating Prioritized 3-DQN and DQN are reasonable as the scheduling algorithm of the composite service model proposed in this paper. However, from the perspective of convergence, our algorithm can obtain a higher cumulative reward value under the same number of training rounds. In addition, When the number of training epochs reaches around 1,600, the Prioritized 3-DQN scheduling algorithm starts to converge. The DQN starts to converge when the number of training rounds reaches about 2,200. Thus, the convergence speed of our algorithm is faster, and a higher cumulative reward value is obtained after the iteration is completed. This is because each time the weight parameters of the neural network are updated in our algorithm, the experience samples with larger time-series differences are selected first, so as to ensure the learning effect of the neural network. From the perspective of stability, Prioritized 3-DQN decouples the selection of the target Q value action and the target Q value calculation, thereby avoiding the problem of overestimation. Therefore, compared with the DQN rising trend, the upward trend of our results is slightly smoother and more stable. In general, our Prioritized 3-DQN is very suitable for composite service scheduling strategies. Compared with DQN, it has higher learning efficiency and can converge earlier to achieve better results.

5.3 Makespan comparison

To verify the generalization ability of Prioritized 3-DQN as a composite service scheduling algorithm, DQN and the four common scheduling algorithms mentioned above are respectively applied to the composite service model. In the process of the comparative experiment, three test sets were used for 20 experiments, the completion time of the composite service was calculated, and the average results were obtained. Figure 2 summarizes the average completion time obtained after 20 experiments on each of the three test data sets.

It can be seen from Figure 2 that the completion time of Prioritized 3-DQN on different test data sets is shorter than the results of the other four scheduling algorithms. Among them, the difference in completion time between DQN and Prioritized 3-DQN is smaller than the other three scheduling algorithms. The completion time of Prioritized 3-DQN on three data sets is about 3.32% less than that of DQN on average. The number of service instances in the three test sets increases sequentially. With the increase in the number of service instances, the increase in the completion time of the composite service under different scheduling algorithms is different, and the gap in completion time between Prioritized 3-DQN and the other four scheduling algorithms is more prominent. This means that the algorithm and DQN algorithm proposed in this paper are more adaptable than other algorithms in terms of completion time.

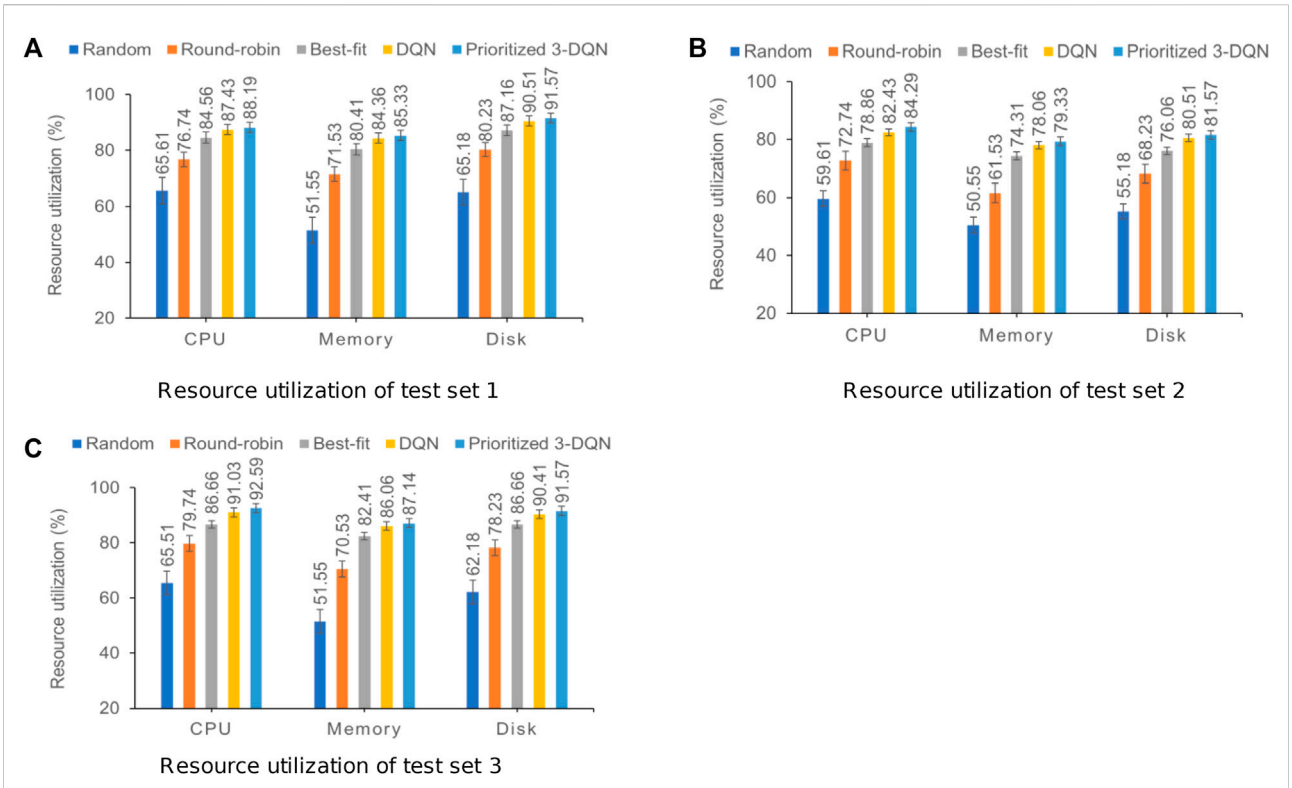

FIGURE 4
Resource utilization results of the three composite service test sets with standard error. **(A)** Resource utilization of test set 1. **(B)** Resource utilization of test set 2. **(C)** Resource utilization of test set 3.

5.4 Quality of service comparison

The degree of user satisfaction is also the main optimization goal of this article. The degree of user satisfaction is closely related to many factors, such as the number of requests for composite services reached per unit time, the number of service instances contained in each composite service, and the processing capacity of the container cluster set in the experiment. In this experiment, five scheduling algorithms are used in the same experimental environment to simulate simulation experiments on three composite service test sets, and then the success rate of each composite service test set is recorded, as shown in Figure 3.

By observing the above graph from a horizontal perspective, our algorithm can achieve the highest success rate compared to other scheduling algorithms. Vertically, with the increase in the number of composite services and service instances, the success rate of each scheduling algorithm after the completion of the composite service allocation is continuously reduced, but the reduction is different. Our algorithm is compared with the other four algorithms. It can be maintained in a relatively stable state, which ensures that the success rate of composite services is about 80% under different composite service test sets. The composite service success rate of Prioritized 3-DQN on the three data sets is about 4.82% higher than that of DQN. From the perspective of diversified loads, the Prioritized 3-DQN is more capable of making reasonable service scheduling decisions than other scheduling algorithms, thereby it increases the success rate of composite services and improves user QoS.

5.5 Resource utilization comparison

In addition to completion time and user QoS, the resource utilization of a container cluster can also be used as one of the criteria for evaluating the performance of scheduling algorithms. This section compares and analyzes resource utilization from the three dimensions: CPU, memory, and disk. During the simulation experiment, the resource utilization rate of the container cluster was recorded after each service instance was scheduled, and the average result of each resource utilization rate was calculated after one round of scheduling was completed. Figure 4 shows the resource utilization results of the three composite service test sets.

The above three graphs show that our prioritized 3-DQN, DQN, and Best-fit algorithms are significantly higher than the

other two algorithms in terms of resource utilization in the three dimensions, indicating that they can make full use of limited resources when scheduling service instances to complete the execution of composite services. When the Best-fit algorithm schedules service instances, it does not consider the data transmission relationship between services and the scheduling of subsequent service instances. It only schedules the current service instance to the container with the best performance and the shortest execution time. Therefore, the resource utilization in the three dimensions is lower than Prioritized 3-DQN and DQN. On the three data sets, the resource utilization of Prioritized 3-DQN on CPU, memory, and disk is about 1.39%, 1.11%, and 1.09% higher than that of DQN, respectively. The Prioritized 3-DQN is also higher than DQN in terms of resource utilization, indicating that Prioritized 3-DQN can make more reasonable scheduling decisions compared to DQN and has a more stable optimization capability under the same environment.

6 Conclusion

Cloud computing has brought great flexibility and cost-effectiveness to end-users and cloud application providers, and it has become a very attractive computing mode for various fields. With the continuous development of biological technology, massive biological data are continuously generated, and the requirements for data processing operation speed, computing power, and stability in practical applications also increase rapidly. Cloud computing has the characteristics of high-speed computing power, high storage capacity, and convenient use, which can meet the needs of biological research. At the same time, cloud providers provide security services to ensure the privacy and integrity of data. When biological samples are processed, each step needs to be supported and completed by cloud services. Between stages, biopharmaceutical companies realize data isolation by transferring data between services. Data quality plays a crucial role in the application effect of data, and the problem of data timeliness is one of the main factors affecting data quality. The timeliness of data can be improved synergistically by combining timeliness rules with statistical technical conditions or functional dependencies. How to use service scheduling strategy to improve service quality and resource utilization has become a key issue in cloud computing. This paper focuses on the core problem of service scheduling management in the container cloud platform. We proposed the composite service model under the modes of container instance (mixed reservation and on-demand), and we proposed the improved DQN algorithm as the scheduling algorithm of the composite service model in this paper. The simulation results show that, under the model presented in this paper, our 3-DQN algorithm is superior to the original DQN algorithm in terms of reliability and convergence. In addition, the algorithm can effectively reduce the completion time of the composite service and improve the user QoS and resource utilization in the container cloud environment.

The method proposed in this paper still has many defects for the actual cloud environment. From the results represented in the paper, the differences in completion time, composite service success rate, and resource utilization between DQN and Prioritized 3-DQN are small. The reason for the smaller difference may be that the scale of our experiments is relatively small. If the scale of the experiments is large, the advantages of Prioritized-3DQN may be more prominent. We also consider comparing Prioritized 3-DQN with the three algorithms used in this paper in the future. In addition, in the process of designing the composite service model in the container cloud environment, the energy consumption and resource cost of the cloud platform are not considered. We can do further research in future work.

Data availability statement

Publicly available datasets were analyzed in this study. This data can be found here: https://github.com/alibaba/clusterdata/blob/v2018/cluster-trace-v2018.

Author contributions

LY, PY, and YD contributed to conception and design of the study. LY wrote the manuscript and performed the statistical analysis. PY and YD helped supervise the project. HQ completed the formatting and editing of the manuscript. All authors contributed to manuscript revision, read, and approved the submitted version.

Funding

NSFC (Nos 61962040, 72062015, and 61662021), Hainan Education Department Project No. Hnky 2019-13, and Hainan University Educational Reform Research Project (Nos HDJY2102 and HDJWJG03).

Acknowledgments

This work was supported by grants from NSFC, Hainan Education Department Project and Hainan University Educational Reform Research Project.

Conflict of interest

The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Publisher's note

References

Almansour, N., and Allah, N. M. (2019). "A survey of scheduling algorithms in cloud computing," in 2019 International Conference on Computer and Information Sciences (ICCIS), Sakaka, Saudi Arabia, 03-04 April 2019, 1–6.

Almezeini, N., and Hafez, A. (2017). Task scheduling in cloud computing using lion optimization algorithm. *Int. J. Adv. Comput. Sci. Appl.* 8 , 78–83. doi:10.14569/ijacsa.2017.081110

Barik, R. K., Lenka, R. K., Rao, K. R., and Ghose, D. (2016). "Performance analysis of virtual machines and containers in cloud computing," in 2016 international conference on computing, communication and automation (iccca), Greater Noida, India, 29-30 April 2016 (IEEE), 1204–1210.

Bernstein, D. (2014). Containers and cloud: From lxc to docker to kubernetes. *IEEE Cloud Comput.* 1, 81–84. doi:10.1109/mcc.2014.51

Chen, Z.-G., Zhan, Z.-H., Lin, Y., Gong, Y.-J., Gu, T.-L., Zhao, F., et al. (2019). Multiobjective cloud workflow scheduling: A multiple populations ant colony system approach. *IEEE Trans. Cybern.* 49, 2912–2926. doi:10.1109/TCYB.2018.2832640

Cheng, M., Li, J., and Nazarian, S. (2018). "Drl-cloud: Deep reinforcement learning-based resource provisioning and task scheduling for cloud service providers," in 2018 23rd Asia and South pacific design automation conference, Jeju, Korea (South), 22-25 January 2018, 129–134. ASP-DAC. IEEE.

Cui, Y., and Xiaoqing, Z. (2018). "Workflow tasks scheduling optimization based on genetic algorithm in clouds," in 2018 IEEE 3rd International Conference on Cloud Computing and Big Data Analysis (ICCCBDA) (IEEE), Chengdu, China, 20-22 April 2018, 6–10.

[Dataset] Alibaba (2018). Cluster-trace-v2018. [EB/OL]. Available at: https://github.com/alibaba/clusterdata/blob/v2018/cluster-trace-v2018 (Accessed December 13,2018).

Dong, T., Xue, F., Xiao, C., and Li, J. (2020). Task scheduling based on deep reinforcement learning in a cloud manufacturing environment. *Concurr. Comput. Pract. Exper.* 32, e5654. doi:10.1002/cpe.5654

Dong, T., Xue, F., Xiao, C., and Zhang, J. (2021). Workflow scheduling based on deep reinforcement learning in the cloud environment. *J. Ambient. Intell. Humaniz. Comput.* 12, 10823–10835. doi:10.1007/s12652-020-02884-1

George, N., Chandrasekaran, K., and Binu, A. (2016). "Optimization-aware scheduling in cloud computing," in Proceedings of the International Conference on Informatics and Analytics, August 25 - 26, 2016, Pondicherry India, 1–5.

Ghasemi, S., Kheyrolahi, A., and Shaltooki, A. A. (2019). Workflow scheduling in cloud environment using firefly optimization algorithm. *JOIV Int. J. Inf. Vis.* 3, 237–242. doi:10.30630/joiv.3.3.266

Islam, M. T., Karunasekera, S., and Buyya, R. (2021). Performance and cost-efficient spark job scheduling based on deep reinforcement learning in cloud computing environments. *IEEE Trans. Parallel Distrib. Syst.* 33, 1695–1710. doi:10.1109/tpds.2021.3124670

Joy, A. M. (2015). "Performance comparison between linux containers and virtual machines," in 2015 International Conference on Advances in Computer Engineering and Applications (IEEE), Ghaziabad, India, 19-20 March 2015, 342–346.

Kyaw, L. Y., and Phyu, S. (2020). "Scheduling methods in hpc system," in 2020 IEEE Conference on Computer Applications (ICCA) (IEEE), Yangon, Myanmar, 27-28 February 2020, 1–6.

Li, F., and Hu, B. (2019). "Deepjs: Job scheduling based on deep reinforcement learning in cloud data center," in Proceedings of the 2019 4th international conference on big data and computing, Guangzhou China, May 10 - 12, 2019, 48–53.

Liang, S., Yang, Z., Jin, F., and Chen, Y. (2020). "Data centers job scheduling with deep reinforcement learning," in *Advances in knowledge discovery and data mining* (New York: Springer International Publishing), 906–917.

Meng, H., Chao, D., Huo, R., Guo, Q., Li, X., and Huang, T. (2019). "Deep reinforcement learning based delay-sensitive task scheduling and resource management algorithm for multi-user mobile-edge computing systems," in Proceedings of the 2019 4th International Conference on Mathematics and Artificial Intelligence, Chegndu China, April 12 - 15, 2019, 66–70.

Myers, T. A., Chanock, S. J., and Machiela, M. J. (2020). Ldlinkr: An r package for rapidly calculating linkage disequilibrium statistics in diverse populations. *Front. Genet.* 11, 157. doi:10.3389/fgene.2020.00157

Orhean, A. I., Pop, F., and Raicu, I. (2018). New scheduling approach using reinforcement learning for heterogeneous distributed systems. *J. Parallel Distributed Comput.* 117, 292–302. doi:10.1016/j.jpdc.2017.05.001

Panwar, N., Negi, S., Rauthan, M. M. S., and Vaisla, K. S. (2019). Topsis–pso inspired non-preemptive tasks scheduling algorithm in cloud environment. *Clust. Comput.* 22, 1379–1396. doi:10.1007/s10586-019-02915-3

Ran, L., Shi, X., and Shang, M. (2019). "Slas-aware online task scheduling based on deep reinforcement learning method in cloud environment," in 2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS) (IEEE), Zhangjiajie, China, 10-12 August 2019, 1518–1525.

Schaul, T., Quan, J., Antonoglou, I., and Silver, D. (2016). "Prioritized experience replay," in International Conference on Learning Representations, San Juan, Puerto Rico, May 2-4, 2016.

Van Hasselt, H., Guez, A., and Silver, D. (2016). "Deep reinforcement learning with double q-learning,", Phoenix, Arizona USA, February 12–17, 2016, 2094–2100. doi:10.1609/aaai.v30i1.10295*Proc. AAAI Conf. Artif. Intell.*30

Wang, Z., Schaul, T., Hessel, M., Hasselt, H., Lanctot, M., and Freitas, N. (2016). "Dueling network architectures for deep reinforcement learning," in International conference on machine learning (PMLR), New York NY USA, June 19 - 24, 2016, 1995–2003.

Wei, Y., Pan, L., Liu, S., Wu, L., and Meng, X. (2018). Drl-scheduling: An intelligent qos-aware job scheduling framework for applications in clouds. *IEEE Access* 6, 55112–55125. doi:10.1109/access.2018.2872674

Xiaoqing, Z., Yajie, H., and Chunlin, A. (2018). "Data-dependent tasks re-scheduling energy efficient algorithm," in 2018 IEEE 4th International Conference on Computer and Communications (ICCC), Chengdu, China, 07-10 December 2018, 2542–2546. IEEE.

Yang, S., Zhu, F., Ling, X., Liu, Q., and Zhao, P. (2021). Intelligent health care: Applications of deep learning in computational medicine. *Front. Genet.* 12, 607471. doi:10.3389/fgene.2021.607471

Zhang, L., Qi, Q., Wang, J., Sun, H., and Liao, J. (2019). "Multi-task deep reinforcement learning for scalable parallel task scheduling," in 2019 IEEE International Conference on Big Data (Big Data), Los Angeles, CA, USA, 09-12 December 2019, 2992–3001. IEEE.