RLPTO: A Reinforcement Learning-Based Performance-Time Optimized Task and Resource Scheduling Mechanism for Distributed Machine Learning

Xiaofeng Lu, *Member, IEEE*, Chao Liu, Senhao Zhu, Yilu Mao, Pietro Lio, and Pan Hui, *Fellow, IEEE*

***Abstract*—With the wide application of deep learning, the amount of data required to train deep learning models is becoming increasingly larger, resulting in an increased training time and higher requirements for computing resources. To improve the throughput of a distributed learning system, task scheduling and resource scheduling are required. This article proposes to combine ARIMA and GRU models to predict the future task volume. In terms of task scheduling, multi-priority task queues are used to divide tasks into different queues according to their priorities to ensure that high-priority tasks can be completed in advance. In terms of resource scheduling, the reinforcement learning method is adopted to manage limited computing resources. The reward function of reinforcement learning is constructed based on the resources occupied by the task, the training time, the accuracy of the model. When a distributed learning model tends to converge, the computing resources of the task are gradually reduced so that they can be allocated to other learning tasks. The results of experiments demonstrate that RLPTO tends to use more compu-ting nodes when facing tasks with large data scale and has good scalability. The distributed learning system reward experiment shows that RLPTO can make the computing cluster get the largest reward.**

***Index Terms*—Cloud computing, scheduling algorithms, dynamic scheduling, resource management, distributed computing, reinforcement learning.**

I. Introduction

DEEP learning is widely used in image identification [1], [2], speech recognition [3], [4], recommendation systems [5], [6], and medical fields [7], [8], [9]. With the development of deep learning research, the performance of deep learning models is becoming increasingly better, but the training time and computing resources required to train deep learning models are also increasing. To meet the computing power requirements of complex deep learning models and the needs of privacy security, distributed machine learning has emerged, in which the training task is jointly completed by multiple distributed computing nodes [10], [11], [12]. Many large IT companies now use several servers and a large number of client or edge devices to form distributed computing clusters, and train on massive data to obtain models with better performance. For example, Google uses the Distributed TensorFlow framework to provide services for Google Photos and Google Cloud Speech [13].

The training of deep learning models requires a large amount of resources and is time-consuming. For example, training a DeepSpeech2 model on the LibriSpeech dataset takes 3-5 days with 16 GPUs to achieve the desired accuracy [14]. Moreover, training a GoogleNet model on the ImageNet dataset takes 23.4 hours on a Titan supercomputing server with 32 NVIDIA K20 GPUs [15].

With the growing scale of distributed computing clusters, determining how to schedule distributed computing clusters to meet the requirements of distributed learning tasks has become the key to the improvement of the throughput of the entire distributed computing cluster [16]. Therefore, it is necessary to design a suitable resource scheduling system for distributed computing clusters and allocate appropriate computing resources for different tasks.

The methods commonly used in the field of resource scheduling include static and dynamic resource scheduling methods. Static resource scheduling involves the allocation of a fixed number of computing resources before task execution, and the computing resources are kept unchanged during task operation. Dynamic resource scheduling is characterized by the dynamic adjustment of the amount of computing resources used by the task during task operation. Classic resource scheduling schemes include first come first served [17] (FCFS), fair scheduling (FS) [18], delayed scheduling [19], and the dominant resource fair (DRF) scheduling policy [20]. These resource scheduling methods do not take advantage of the fact that the performance

Manuscript received 28 October 2022; revised 1 September 2023; accepted 12 September 2023. Date of publication 20 September 2023; date of current version 30 October 2023. This research work was supported by the National Key R&D Program of China under Grant 2020YFB2104700, and in part by the National Natural Science Foundation of China under Grant 62136006. Recommended for acceptance by Alan Sussman. *(Corresponding authors: Xiaofeng Lu.)*

Xiaofeng Lu, Chao Liu, and Senhao Zhu are with the National Engineering Center for Mobile Internet Security Technology, Beijing University of Post and Telecommunications, Beijing 100876, China (e-mail: luxf@bupt.edu.cn; liuchao37@yeah.net; zsh239040@bupt.edu.cn).

Yilu Mao is with the Alibaba Cloud Computing Company Ltd., Beijing 100102, China (e-mail: yilu.myl@alibaba-inc.com).

Pietro Lio is with Computer Laboratory, University of Cambridge, CB2 1TN Cambridge, U.K. (e-mail: pl219@cam.ac.uk; panhui@ust.hk).

Pan Hui is with the Computational Media and Arts Thrust, Hong Kong University of Science and Technology, Guangzhou 510000, China (e-mail: pan.hui@telekom.de).

Digital Object Identifier 10.1109/TPDS.2023.3317388

of the deep learning model changes with the training time, and do not maximize the throughput of the entire distributed computing cluster.

Reinforcement learning is a research hotspot in machine learning. Its main idea is what actions an agent should take to maximize the reward in a particular environment. Reinforcement learning is a type of unsupervised learning where the agent learns from past experience without labeled data. Resources and tasks in distributed machine learning clusters are non-linear and non-deterministic, and reinforcement learning can adapt to these rapid changes in the environment. Reinforcement learning can deal with long-term rewards, adapting to situations where resource scheduling decisions pay off in the future. It is suitable for complex systems such as distributed learning clusters. In this paper, deep reinforcement learning is used for resource scheduling in distributed learning clusters, the strong fitting ability of deep learning is used to fit the complex environment in distributed systems, and the decision-making ability of reinforcement learning is used to decide which resource scheduling behavior to adopt.

The main contributions of this research are as follows.

1) A computing resource allocation algorithm based on long-term and short-term task volume prediction, namely LSP, is proposed. The LSP prediction method combining long- and short-term prediction is proposed to predict the future task volume. The index moving weighted average method is used to evaluate the number of tasks completed by each computing node and to calculate the number of computing nodes required for a certain day. In this way, the energy consumption of a distributed computing cluster can be reduced on the premise of ensuring relatively sufficient computing resources.
2) A task scheduling method based on a multi-priority task queue is used. This research is oriented to the distributed machine learning, and the dynamic priority and preemptive task scheduling mechanism is used to place the learning tasks into task queues with different priorities. High-priority tasks can obtain computing resources first to start training tasks earlier.
3) A dynamic computing resources scheduling algorithm (RLPTO) based on reinforcement learning is proposed in this paper. The reward function and state space of the reinforcement learning algorithm are designed according to the model accuracy, training time and the task priority. The reinforcement learning algorithm dynamically increases and reduces the computing resources of tasks. RLPTO combines resource scheduling with task scheduling to maximize the reward of distributed computing clusters. This paper finds the relationship between β and data scale, and gives a suggested range of β.

II. Related Work

Increasingly more researchers are investigating how to reasonably schedule computing resources in distributed clusters to different learning tasks to improve the utilization of computing resources.

A. Resource Scheduling Strategy

Common centralized scheduling schemes in the field of resource scheduling include FCFS [17], FS [18], delayed scheduling [19], and DRF scheduling [20].

FCFS [17] is the most basic resource scheduling strategy. The operating system allocates resources to tasks according to the time sequence of task submission, and each task uses the allocated resources to work.

The FS strategy [18] implements resource scheduling based on queues. During resource allocation, the queues are first allocated the minimum amount of resources, and the remaining resources are then allocated to each queue according to the resource shortage of different queues. Within each queue, resources are allocated to all jobs in an evenly divided manner.

The purpose of delayed scheduling [19] is to increase the data locality of task scheduling. For the task at the head of the queue, the scheduler determines whether the data used by the task are local. If the data are localized data, the task can be executed. Otherwise, the task scheduler executes other tasks with local data behind the queue. When the task at the head of the line is delayed for a certain period of time, the task is started.

DRF scheduling [20] is a max-min algorithm, and the scheduling goal of the strategy is to maximize the minimum amount of resources allocated to a task. The resource that the task requires the most is called the dominant resource. The purpose of DRF is to keep the dominant resources of different tasks as fair as possible.

B. Resource Scheduling for Distributed Learning

There are different optimization methods for resource scheduling in distributed learning. Yan et al. established a resource-completion time model [21]. The model evaluates the impacts of data parallelism, model parallelism, and different resource scheduling methods on tasks in distributed machine learning to obtain the optimal resource allocation scheme. Lee et al. [22] proposed a dynamic resource management scheme for distributed machine learning with a federated learning parameter server structure. The method evaluates each iteration time according to the computation time and communication time, based on which it continuously adjusts the resource allocation. This method addresses the dynamic demands on the resources of distributed machine learning tasks during execution.

Some researchers have taken a more detailed look at the use of GPUs for machine learning tasks. One problem of distributed machine learning is that CPU computing is slower than GPU computing, but GPU training requires more network bandwidth than CPU training. To solve this problem, Gao et al. [23] proposed GAI, a scheduler based on a centralized tree structure, the scheduling objective of which is to minimize the completion time of high-priority training jobs. GAI uses a centralized rack-aware tree scheduling method, and maintains a resource tree in

the memory to place all tasks of the machine learning training jobs in one machine, or in the machines belonging to the same rack to the greatest extent possible.

Gu et al. [24] proposed two GPU scheduling algorithms, namely the Discretized Two-Dimensional Gittins index algorithm and the Discretized Two-Dimensional LAS algorithm, to minimize the average job completion time. According to different task priorities, GPU resources are allocated and scheduled to reduce the task waiting time, thereby reducing the completion time of all learning tasks.

In addition to static resource scheduling schemes for distributed machine learning, dynamic scheduling schemes are also evolving. Peng et al. proposed the Optimus algorithm [25], which dynamically adjusts the number of worker nodes managed by the parameter server during runtime. The algorithm fits the convergence of the machine learning model online, and uses the constructed performance model to evaluate the number of resources required for each job, thereby achieving the goal of minimizing the total task completion time. However, it is quite difficult to fit the convergence of the machine learning model online, especially for users who only use data to train models.

In addition to modeling the relationship between the resources and the task completion time, some research has aimed to maximize the overall performance of multiple models on a cluster. Zhang et al. [26] proposed the SLAQ algorithm, which models the task execution quality-runtime while carrying out resource allocation. SLAQ allocates resources according to the change of the current loss value of different models. With the assumptions of the loss convergence rate, SLAQ uses the historic values of exponentially weighted loss to fit a curve for sublinear algorithms. Due to the differences in the structure of each deep learning model, and because the computing resources occupied by the models can change at any time, methods that predict the model quality based on mathematics sometimes lead to large prediction errors.

Zheng et al. [27] proposed a target-based resource scheduling scheme, TRADL, which aims to improve the overall performance of multiple distributed deep learning tasks. This method sets up two-layer goals for distributed learning tasks. When the accuracy of the model reaches a goal, the model is handed over to the user, and the model continues to be trained to improve the model accuracy. This scheme reduces the time required for the model to reach a target. While TRADL is aimed at model accuracy, users usually want a model with high accuracy, and it is difficult for users to set the target accuracy at the beginning.

It is not easy to change the configuration of a distributed machine learning system when it is running. Chun et al. [28] proposed Dolphin to address the configuration problem at runtime by optimizing the configuration of the running machine learning system according to computations and communications. Dolphin extends the existing parameter server architecture with two new components, namely the Optimizer and Elastic Memory Store (EMS). The Optimizer models the iteration time as a cost function of several variables. Via cost-based optimization, the Optimizer finds the optimal configuration that produces the lowest cost, i.e., the shortest iteration time. EMS is a distributed in-memory store abstraction that makes it possible to change the configuration of the system at runtime. However, Dolphin only focuses on the iteration time and other time durations, not on the performance of the model. The proposed approach focuses on the cost of improving model performance.

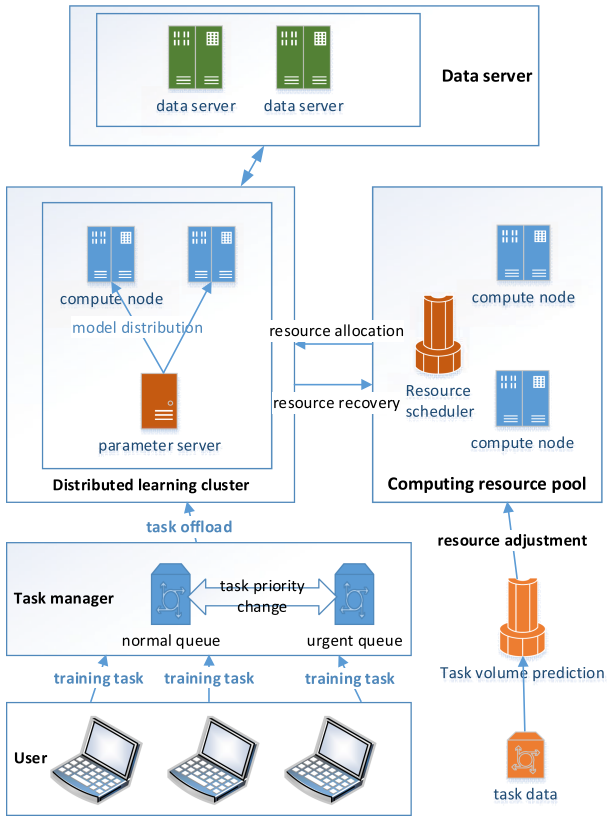

Fig. 1. Overall structure of the proposed system.

III. Task and Resource Scheduling System Of Distributed Machine Learning

The distributed machine learning resource allocation and scheduling algorithm based on task prediction and reinforcement learning proposed in this work includes three components: resource allocation, task scheduling, and resource scheduling. The overall structure of the system is illustrated in Fig. 1.

The user initiates a learning task request, and the task will enter the task manager. The task manager schedules tasks according to the priority of tasks, and assigns computing resources to the selected tasks. The dynamic priority task scheduling mechanism is adopted for the task manager. Higher-priority tasks can obtain computing resources first to complete training tasks earlier. The priority of a task is set and adjusted by the user, and the user can send a request to the task manager to change the priority of the task. The priority of the task is determined by the cost paid by the user; high-priority tasks cost more money. Therefore, the user will not define all tasks as high-priority.

The resource manager predicts the daily task volume and determines the size of the resource pool available for the day. In this work, a task prediction algorithm called LSP is proposed.

LSP combines the advantages of the autoregressive integrated moving average (ARIMA) and gated recurrent unit (GRU) models to predict future tasks. The resource manager calculates the daily computing resources via the exponentially weighted moving average (EWMA) method and the LSP task volume prediction model.

After computing resources are allocated to the task, the learning task begins to execute. In the training process of the distributed machine learning model, the resource scheduler schedules the computing resources occupied by each learning task based on the reinforcement learning algorithm. After the machine learning model has been trained for a long amount of time and the performance of the model reaches convergence, the resource scheduler actively reduces the amount of computing resources occupied by the task.

IV. Computing Resource Allocation Based on Long- and Short-period Task Volume Prediction

A. Prediction of Task Volume Based on Long and Short Periods

The daily task volume of the system is affected by the task volume in the recent period, and it also exhibits long-term periodicity, i.e., people do similar work every week. Therefore, the amount of data processed in the previous weeks can be used to predict the amount of data in the future. For example, to predict the task volume of next Monday, the task volume of each Monday in the past few weeks can be used to make predictions.

This study proposes the combination of the GRU and ARIMA models for task volume prediction. The GRU model is used to make predictions based on the task volume of the last week, and the ARIMA model is used to predict the task volume of the next week based on the data volume of the previous weeks.

The GRU is a type of recurrent neural network. Like long short-term memory, (LSTM), it was also proposed to solve problems such as long-term memory and gradients in backpropagation.

The ARIMA model is composed of an autoregressive (AR) model and a moving average (MA) model. AR models are mainly used to describe the relationship between the current moment and the historical moment. The AR model fits the historical data and predicts the data at the current moment according to the fitting result. The MA model pays more attention to the influence of the error term. The role of the MA model in the ARIMA model is primarily to adjust the error term in the model.

This work combines the GRU and ARIMA models, and proposes a long- and short-term combined prediction algorithm called LSP. The prediction formula is

$$y_{pred} = \alpha GRU\left(x_{before\_seven}\right) + (1-\alpha)\, ARIMA\left(x_{before}\right), \tag{1}$$

where α is the scale factor, the value range of which is [0, [1], representing the influence of different models on the final predicted value. If the data to be predicted is stable over the long term, the α value can be set closer to 0, and the output will be more inclined to the ARIMA model. If the data to be predicted changes significantly over time, the α value is set closer to 1, and the output is more skewed toward the GRU model. In this study, compared with the long-term task volume, the short-term task volume plays a greater role in predicting the future task volume, so the scale factor α is taken as 0.6.

TABLE I
Notations and the Representation

α	the scale factor of LSP
ρ	the scale factor of EWMA
μ	the priority change factor
N	the number of user requests to change the priority within a priority update period
va_t^i	the variation of the model accuracy of the i-th task at time t
T^i	the execution time of the i-th task
w^i	the number of computing nodes owned by the i-th task
kw	the average power consumption of the computing nodes in each computing cycle
pr_t	the priority of the current task at the t priority update period.
va_t^i	the variation of the model accuracy of the i-th task at time t
β_0	model accuracy variation threshold
γ^t	The discount factor at time t
θ	Actor network parameter of PPO
ω	Critic network parameter of PPO
$A^{\pi_\theta}(s,a)$	the advantage function
$Q_\omega(s,a)$	the action value function with ω as parameter
π	policy function
$Q^{\pi_\theta}(s,a)$	the action value function corresponding to policy function
$V^{\pi_\theta}(s)$	the environmental value function corresponding to the policy function
∇_θ	derivation of θ
δ^{π_θ}	error generated by the policy function
r	cumulative reward value
$J(\theta)$	the overall reward function
$P(X) \parallel Q(X)$	relative entropy of $P(X)$ and $Q(X)$
β_{kl}	the adaptive KL divergence constraint

Table I refers to the notations and their representations used throughout this paper.

B. Resource Management Based on Task Volume Prediction

To reduce the energy consumption and maintenance costs, usually only some computing resources are powered on. However, it is difficult to determine how many computing resources should be powered on.

In this study, a heuristic algorithm is used to manage the number of running nodes in a distributed learning cluster. If the predicted number of running nodes is greater than the current number of running nodes, the resource manager increases the

running nodes. When the predicted number of running nodes is less than the current number of running nodes, the number of running nodes will be reduced, thereby reducing the operation and maintenance costs.

The EWMA (Exponentially Weighted Moving Average) method is used in this study to calculate the average number of tasks completed by nodes. When using this method, each previous value decreases exponentially with time. The formula for the average daily number of tasks completed by a computing node is

$$task_{new} = \rho task_{old} + (1 - \rho)\, task_{past}, \tag{2}$$

where ρ is a scale factor, which represents the impact of past data on the average value, $task_{past}$ represents the task volume in the past day, $task_{old}$ represents the task volume calculated by the EWMA the last time, and $task_{new}$ represents the task volume calculated at the current moment. If the number of tasks changes frequently, set ρ to close to 0. If the number of tasks changes infrequently, set ρ to close to 1. In this study, $\rho = 0.8$.

Using (1) and (2), the number of running computing nodes required by the distributed computing cluster on the current day can be calculated. The calculation formula is as follows.

$$w_{pred} = \frac{y_{pred}}{task_{new}} \tag{3}$$

V. Task Scheduling Based on Multi-Priority Queues

A. Dynamic Priority

The types of machine learning tasks proposed by users are diverse, as are the time requirements for completing the tasks. In this work, tasks are divided into two types, namely normal tasks and urgent tasks, and tasks of each type have their own priorities. Users prioritize their tasks according to their needs. The task scheduler schedules tasks according to the priority of the task; high-priority tasks are executed earlier, while lower-priority tasks must wait longer.

The priority of tasks is defined by the user and can be adjusted dynamically. When a user wants his or her task to be completed first, the user can increase the priority of the task. In a resource scheduling cycle, the user can increase the priority of the task multiple times. The priority adjustment formula is

$$pr_t = pr_{t-1} + \mu N \tag{4}$$

where pr_{t-1} represents the priority of the current task, pr_t is the priority after adjustment, t is the priority update period, and μ represents the change factor, which represents the amount of change in each request to increase or decrease the priority. The larger the value of μ, the greater the impact of each user request on the task. The value of μ is set to 0.1 in this study. Finally, N represents the number of user requests to change the priority within a priority update period.

To facilitate the management of task queues, a threshold is set for task priority. When the priority of a normal task is greater than the threshold, the task is set as an urgent task and the priority is reset. When the priority of an urgent task is lower than the threshold, the urgent task is downgraded to a normal task and a new priority is set.

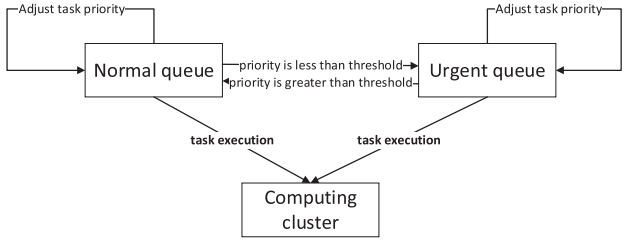

Fig. 2. Conversion of tasks between the normal queue and emergency queue.

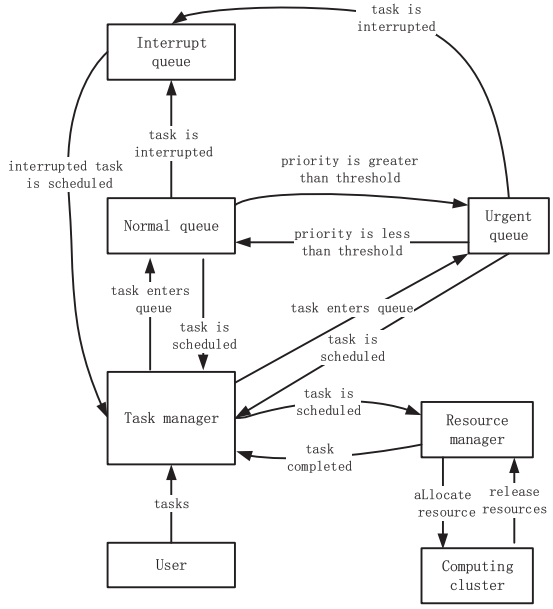

Fig. 3. Task scheduling mechanism of RLPTO.

B. Task Scheduling Based on Multi-Priority Queues

To meet the scheduling requirements of tasks with different priority, multi-priority queues are used for task scheduling. The task manager establishes three queues, namely the normal queue, urgent queue, and interrupt queue. The normal queue includes low-priority and default-priority tasks. The urgent queue mainly includes urgent tasks with high priority. The tasks in this queue must be executed earlier than the tasks in the normal queue. The conversion of tasks between the normal queue and emergency queue is shown in Fig. 2.

The interrupt queue is used to store interrupted tasks. In the proposed task scheduling scheme, there is a task with the highest priority, and this task occupies resources in a preemptive manner for task execution. The interrupt queue saves the interrupted tasks, and the tasks in the queue will also be scheduled preferentially in the subsequent task scheduling. The proposed task scheduling mechanism of distributed machine learning is presented in Fig. 3.

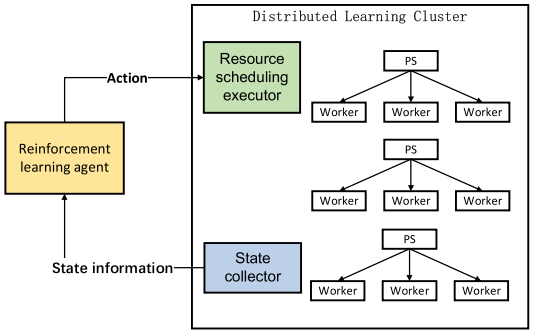

Fig. 4. Architecture of the distributed resource scheduling system.

In the distributed task scheduling system, the task scheduler selects tasks from multiple queues and offloads them to the computing cluster. When a task in the computing cluster is completed, the task manager first checks to see if there are still tasks in the interrupt queue. If there is a task in the interrupt queue, the task manager selects the task from the interrupt queue to execute first; otherwise, it selects the task with higher priority from the urgent queue to execute. If the urgent queue has no tasks, the task manager schedules high-priority tasks from the normal queue for execution. When the highest-priority task appears and there are no idle computing resources in the computing cluster, the newly-started learning task is interrupted. The interrupted task is added to the interrupt queue, and its resources are occupied by the task with the highest priority. For each task in the task queue, the owner of the task can adjust the task priority or cancel the task.

VI. Resource Scheduling Algorithm for Distributed Learning Systems Based on Reinforcement Learning

A. Resource Scheduling System Architecture for Distributed Learning Based on Reinforcement Learning

In this work, a distributed resource scheduling algorithm based on deep reinforcement learning is used for the resource scheduling of machine learning tasks. The distributed resource scheduler consists of a state collector, reinforcement learning agent, and resource scheduling executor. The state collector collects and processes the state information of the ongoing distributed learning task, and inputs the processed state information into the reinforcement learning agent. The reinforcement learning agent chooses the next action to take based on the input state, and sends the action to the resource scheduling executor. The resource scheduling executor performs this action and adjusts the amount of resources owned by each parameter server of distributed machine learning. The strong fitting ability of deep learning is used to fit the complex environment in the distributed system, and the decision-making ability of reinforcement learning is used to decide the resource scheduling operation to be taken.

The architecture of the distributed resource scheduling system is exhibited in Fig. 4.

Resource Scheduling Algorithm.

```
1.  function Schedule(Environment env, Agent agent):
2.  reduce task' resource;
3.  cur_state = env.getState(); //Get the environ mental
    state
4.  While ThereIsFreeResource() = True do:
5.    action = agent.getAction(cur_state); // Get the next
      action by Reinforcement learning algorithm
6.      cur_state = env.doAction(action); // Execute
        actions and update state
7.  end function
9.   function main():
10. Initialize environment;
11. Initialize Task Queue;
12. Initialize agent;
13. While true do:
14.   If TaskQueue.empty() do:
15.       Continue;
16.   End if
17.   If model is convergent or server don't train model
      do:
18.       If model is convergent do: // The model doesn't
          need any more training
19.         Release resources;
20.       End if
21.       task = getTask(); // Take a new task from the
          task queue
22.       AllocateResource(task);
23.       StartTrain(task);
24.   End if
25.   If task's precision increase < β do:
26.       Schedule(environment, agent); // Reduce some
          resources
27.       End if
28.   End while
29. End function
```

As shown in Fig. 4, the reinforcement learning agent selects the next action based on the training state of the machine learning model being trained. Distributed machine learning requires a large amount of iterative training at the beginning of the task, and reinforcement learning agents should not frequently change the computing resources at this time. When the machine learning model is close to convergence or has converged, and once the gradient of the model parameters has changed very little after each training iteration, the resources allocated to the task can be modified. This work proposes the use of a reinforcement learning algorithm to adjust the computing resources of this task, and to allocate a portion of the resources of the task to other tasks. In this way, the computing resources occupied by the converged distributed learning tasks can be released in time, and the excessive use of computing resources by such tasks can be reduced, thereby improving resource utilization. The resource scheduling algorithm is given as follows.

B. Reward Function for RLPTO

The goal of the resource scheduling strategy proposed in this paper is to maximize the utilization of computing resources and improve the training speed of each distributed learning task under the premise of achieving the high performance of the distributed learning model. To achieve this goal, it is necessary to provide an accurate state space and an action space with low computational complexity, as well as to design a reasonable reward function for the reinforcement learning algorithm.

When users submit a machine learning task, they usually desire that the learning task be completed as soon as possible so that the performance of the model can be known as early as possible. Thus, the training effect and resource utilization efficiency of the distributed learning model are comprehensively considered, and a reinforcement learning-based performance/time-optimized (RLPTO) resource scheduling algorithm is proposed.

The reinforcement learning algorithm evaluates the current impact of the action a_t at time *t* via the reward function. When the performance of the model is improved, the reward function has a positive benefit, and it is necessary for the task to occupy the computing node. When the model tends to converge and stabilize, although the task continues to occupy computing resources, the performance of the model cannot be improved. In this case, the occupied computing resources should be gradually reduced, and the released computing resources can be allocated to other training tasks that require computing resources. This study proposes a reinforcement learning reward function that combines the model performance and training time. The reward function is defined as follows:

$$\begin{aligned} va_t^i &= acc_{new}^i - acc_{pre}^i \\ r_t^i &= va_t^i + sign\left(va_t^i - \beta_0\right) \times T^i \times w^i \times kw, \end{aligned} \tag{5}$$

where va_t^i represents the variation of the model accuracy of the *i*-th task at time *t*, β_0 represents the threshold of the variation of the model accuracy, T^i represents the execution time of the *i*-th task, w^i represents the number of computing nodes owned by the *i*-th task, and *kw* is a constant that represents the average power consumption of the computing nodes in each computing cycle, and $kw = 0.000625$. *sign* is a function of judging whether $(va_t^i - \beta_0)$ is positive or negative. In the early stage of model training, the accuracy of the model continues to improve, and $sign(va_t^i - \beta_0) > 0$. After the model has been trained for a period of time, the performance of the model has converged, and the change in the accuracy of the model is very small, $sign(va_t^i - \beta_0) < 0$. When $va_t^i > \beta$, the training of the model for the *i*-th task brings a positive benefit. When $va_t^i < \beta$, with the increase of the task execution time, the use of computing nodes for the training of the *i*-th task will bring negative benefits, i.e., computing resources may be wasted.

The goal of reinforcement learning decision-making is to obtain the maximum cumulative reward. Two commonly used methods for calculating the cumulative reward are the *T*-step and gamma discount cumulative reward methods. The *T*-step cumulative reward method calculates the total cumulative reward value in *T* decision-making cycles, and then takes the average value. The calculation formula is

$$reward = E\left[\sum_{i=1}^{T-1} r_i\right] \tag{6}$$

where *T* is the number of reinforcement learning steps from the start to the current time. The γ discount cumulative reward discounts the reward value of the previous strategy when calculating the current cumulative reward. The calculation formula is

$$reward = E\left[\sum_{t=1}^{\infty} \gamma^t r_t\right] \tag{7}$$

where γ^t is the discount factor at time *t*, $\gamma \in (01)$. The closer the value of γ is to 0, the smaller the effect of the previous strategy; the closer the value of γ is to 1, the more important the previous strategy. The gamma discount cumulative reward method is adopted to calculate the total reward. In this study, resource scheduling is progressive, and each task increases or decreases by at most one computing resource in each scheduling cycle. Therefore, the previous strategy is very important for the calculation of the cumulative reward, so $\gamma = 0.99$.

C. Proximal Policy Optimization Reinforcement Learning Algorithm

The policy gradient (PG) algorithm is an important category in deep reinforcement learning. The PG algorithm models the policy function and then uses gradient descent to update the parameters of the network. One disadvantage of the PG method is that the parameter update is slow. For this reason, researchers have proposed the combination of the value-based and policy-based algorithms to form the actor-critic (AC) algorithm. The actor part of the algorithm is responsible for updating the policy function π_θ and selecting subsequent actions. The critic part is responsible for calculating the action value $Q_w(s,a)$, and gives the score for the action chosen by the actor. s means the state and a means the action. The actor modifies the probability of all available actions based on the rating given by the critic. The update of the network parameter θ by the actor depends on the action-value function of the critical part. The critic part updates its network parameters ω by function approximation. The AC algorithm updates the gradient in a way similar to the PG algorithm. The overall reward function $J(\theta)$ is defined as follows:

$$\nabla_\theta J(\theta) \approx E_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\, Q_\omega(s,a)\right] \tag{8}$$

where π is the policy function. Because the actor part is a policy-based algorithm, there is a problem of medium to high variance. In addition to introducing the critic network to reduce the variance of the actor, the baseline method is also introduced to further reduce the variance. The AC algorithm introduces the advantage function $A^{\pi_\theta}(s,a)$ as the baseline function. The meanings of $A^{\pi_\theta}(s,a)$, $Q_w(s,a)$, $Q^{\pi_\theta}(s,a)$, $V^{\pi_\theta}(s)$, ∇_θ, δ^{π_θ}, $J(\theta)$, β_{kl} can be found in Table I. The policy gradient update formula is given as follows.

$$\nabla_\theta J(\theta) \approx E_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\, A^{\pi_\theta}(s,a)\right], \tag{9}$$

The advantage function is defined as

$$A^{\pi_\theta}(s,a) = Q^{\pi_\theta}(s,a) - V^{\pi_\theta}(s), \tag{10}$$

where $Q^{\pi_\theta}(s,a)$ is the action value function corresponding to the strategy function, and $V^{\pi_\theta}(s)$ is the environmental value function corresponding to the policy function. This function calculates the improvement compared to the average of the action taken at that state. In other words, this function calculates the extra reward if this action is taken. The extra reward is that beyond the expected value of that state. The advantage function $A^{\pi_\theta}(s,a)$ uses a *T*-step update to estimate the advantage. The AC algorithm uses the temporal-difference (TD) error method to estimate $A^{\pi_\theta}(s,a)$ unbiasedly. The calculation formula of the TD error is

$$\delta^{\pi_\theta} = r + \gamma V^{\pi_\theta}(s') - V^{\pi_\theta}(s), \tag{11}$$

where *r* is the accumulated reward value. By combining (9) and (11), the GP update method of the AC algorithm can be obtained as follows:

$$\nabla_\theta J(\theta) \approx E_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\,\delta^{\pi_\theta}\right], \tag{12}$$

where δ^{π_θ} is the error generated by the policy function. The actor network part uses the PG algorithm for action selection, but the PG algorithm requires resampling after each update.

On the basis of the AC algorithm, the PPO algorithm is proposed to include the addition of importance sampling, which turns the online algorithm into an offline method so that the agent can reuse the previously sampled data. In the traditional AC algorithm, the sampling method is as follows.

$$E_{x\sim p}[f(x)] = \int f(x)\,p(x)\,dx \approx \frac{1}{N}\sum_{i=1}^{N} f\left(x^i\right), \tag{13}$$

The principle of importance sampling is as follows.

$$\begin{aligned} E_{x\sim p}[f(x)] &\approx \int f(x)\,p(x)\,dx = \int f(x)\,\frac{p(x)}{q(x)}dx \\ &= E_{x\sim p}\left[f(x)\,\frac{p(x)}{q(x)}\right], \end{aligned} \tag{14}$$

The importance principle is introduced into the PC, so the PG is as follows.

$$\nabla_{\theta'} J(\theta) = E_{\pi_{\theta'}}\left[\frac{\pi_\theta(s,a)}{\pi_{\theta'}(s,a)} A^{\pi_{\theta'}}(s,a)\right] \tag{15}$$

θ' is the new network parameters. To ensure that the difference between θ and θ' is not too large, the Kullback-Leibler (KL) divergence is added to the PPO algorithm. The KL divergence is used to measure the "distance" between two probability distribution functions, and its expression is given by the following equation.

$$\begin{aligned} KL[P(X) \parallel Q(X)] &= \sum_{x\in X}\left[P(x)\log\frac{P(a)}{Q(x)}\right] \\ &= \sum_{x\in X}\left[P(x)\log\frac{P(a)}{Q(x)}\right] \end{aligned} \tag{16}$$

PPO Algorithm.

1: begin
2: Initial policy function parameters θ_0, and value function parameters w_0
3: for $i \in \{1,\ldots,N\}$ ***do***
4: Using θ_k to interact with the environment to collect $\{s_t, a_t\}$
5: Computing advantage function $A^{\theta^k}(s_t,a_t)$
6: Find policy function parameters θ to optimize $J_{PPO}(\theta)$
7: end for

The KL divergence and importance sampling together affect the training speed and accuracy of the AC algorithm. The final likelihood function is

$$J^{\theta^k}_{PPO}(\theta) = J^{\theta^k}_{PPO}(\theta) - \beta_{kl} KL(\theta,\theta'), \tag{17}$$

where β_{kl} is the adaptive KL divergence constraint.

The PPO algorithm is as follows.

D. Reinforcement Learning State Space

The learning process of reinforcement learning can be expressed as Markov decision processes (MDPs). Assuming that the state space of the environment E where the agent is located is S, all states $s_t \in S$ in the environment E can be perceived by the agent, and the set of all actions a of the agent is the action space A. At a certain time t, the agent obtains the state s_t of its environment, performs an action a_t, affects the current state s_t, and changes the current state s_t to another state s_{t+1} via the state transition function P. The change of state will cause the reward function R to feed back the corresponding reward r_t to the agent. By continuously selecting the next action a $\in$ A and obtaining the corresponding reward r, the current environment finally reaches the terminal state.

For the agent to train a better policy function π, it is necessary to provide the agent with as much accurate and useful environmental information as possible. State information can accurately describe various important information in a distributed machine learning cluster, including not only the execution speed and model performance of distributed learning tasks, but also related information such as the resource allocation of tasks in the distributed cluster.

In this work, N_t is used to denote the number of learning tasks performed in the distributed learning cluster at the t-th sampling. By adjusting the number of learning nodes for each distributed learning task, the model performance and training time of multiple distributed learning tasks are balanced. The state information obtained at the t-th sampling can be described as $s_t = (\vec{m_t}, \vec{e_t}, \vec{l_t}, \vec{n_t}, \vec{a_t}, \vec{c_t}, \vec{w_t})$.

$\vec{m_t}$ is an N-dimensional vector representing N tasks being performed in the distributed learning cluster at the t-th sampling. $\vec{e_t}$ is an N-dimensional vector representing the number of iterations that the corresponding task has completed at the t-th sampling. $\vec{l_t}$ is an N-dimensional vector representing the

loss function value of each task in the current iteration cycle at the t-th sampling. $\vec{n_t}$ is an N-dimensional vector representing the maximum loss function value of each task until the t-th sampling. $\vec{a_t}$ is an N-dimensional vector representing the model accuracy of N tasks in the current iteration cycle at the t-th sampling. $\vec{c_t}$ is an N-dimensional vector representing the usage time of the computing nodes by N tasks in the current iteration cycle at the t-th sampling. $\vec{w_t}$ is an N-dimensional vector representing the proportion of computing nodes owned by each task in the current iteration cycle at the t-th sampling.

E. Reinforcement Learning Action Spaces

Reinforcement learning agents take actions based on the collected environmental states. When using reinforcement learning for the resource scheduling of distributed learning tasks, the reinforcement learning agent must obtain the model state information of all running tasks, and the policy function then selects the next action to execute. The resource scheduler performs this action to schedule computing resources. Therefore, the period of action change is the maximum value among the training iteration periods of these tasks.

In reinforcement learning, different actions have different effects on the environment in which the agent is. Suppose the two-dimensional action space in a distributed environment is [*N, M*]; then, the selection range of the action space is *N*∗*M*. When the values of *N* and *M* are both large, the number of actions that the reinforcement learning agent must learn is very large, which increases the training cost of reinforcement learning and leads to a significant decrease in the convergence speed.

A simple action strategy is therefore designed: the agent's next action directly increases or decreases the number of computing nodes assigned to a task. The action is defined as $a_t = i$, and its specific meaning is as follows: when $i > 0$, the number of computing nodes for the *i*-th task is increased; when $i < 0$, the number of computing nodes for the *i*-th task is reduced; when $i = 0$, the agent does not change the number of computing nodes.

VII. Experiments and Results

A. Experimental Setup

1) Experimental Environment: The server used in this experiment had 48 GB of memory, two 1080Ti graphics cards, and a 2.5 TB hard disk. The CPU was an Intel Core^{TM} i7-8700K, and the operating system was Ubuntu 18.04.6 LTS. The Docker version used in this research was 20.10.10.

Based on the latest version of the image released by TensorFlow, the container image of the distributed learning system was constructed, and the image was used as the bottom layer to build computing nodes to form a distributed computing cluster.

2) Experimental Data: There were two types of distributed learning task in this experiment: convolutional neural network (CNN)-based image classification, residual neural network (ResNet)-based image classification.

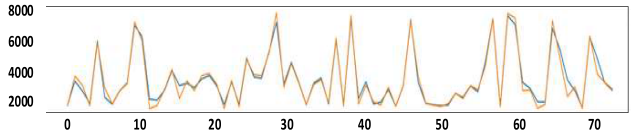

Fig. 5. Prediction results of the proposed LSP method.

In order to investigate the scalability of the proposed method on datasets of different scales, this experiment used image datasets of different scales, 40000, 60000, and 120000 respectively. In the following text, they are defined as *small-scale data, medium-scale data, and big-scale data*. Although 120000 images are not considered a large scale in many systems, due to the hardware condition limitations of our server, more data can significantly increase the training time, making it difficult to conduct the experiments.

In order to investigate the scalability of the proposed method on different scale computing nodes, experiments were conducted on different computing node scales, namely 8, 12, and 16.

When training a distributed learning model, there are two conditions for ending the task, namely that the specified number of iterations has been reached or the model has converged. The specified number of iterations refers to the number of training times for the learning task preset by the user. Once the model reaches the convergence state, the accuracy of the model has exhibited little improvement and the loss function value has converged to a certain fixed value.

B. Experimental Results

1) Task Volume Prediction and Resource Allocation: The task volume of distributed learning can be represented by the size of the dataset used by the distributed learning model. In this research, the traffic data of the core network of a city in Europe were used as the training data to test the prediction algorithm. This dataset was collected from the Internet traffic data of a city in Europe from 06:57 on June 7, 2005, to 11:17 on July 31, 2005 [29]. The prediction results are shown in Fig. 5.

In Fig. 5, the orange line is the actual task value, and the blue line is the prediction result of the proposed LSP method, which achieved an accuracy rate of 97.1%. Although Internet traffic is not the number of machine learning tasks, this paper argues that they all represent the cyclical nature of how humans use networks and computers. The experimental results show that the method can be used for task volume prediction and can maintain a high model accuracy.

In the case of the same number of tasks, different numbers of running computing nodes will impose different computing pressures on each computing node. Suppose the number of computing nodes is 60. Based on the static resource allocation method, 30 computing nodes were used. Fig. 6 shows the average number of tasks that each computing node needs to complete based on the proposed resource allocation method and the static resource allocation method. As can be seen from the figure, the proposed resource allocation scheme significantly reduced the number of tasks completed by each computing node in all cycles.

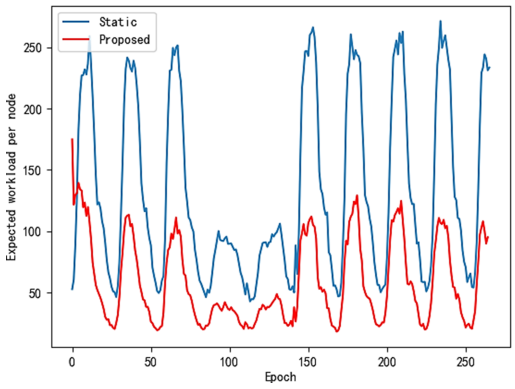

Fig. 6. Average number of tasks completed by each computing node.

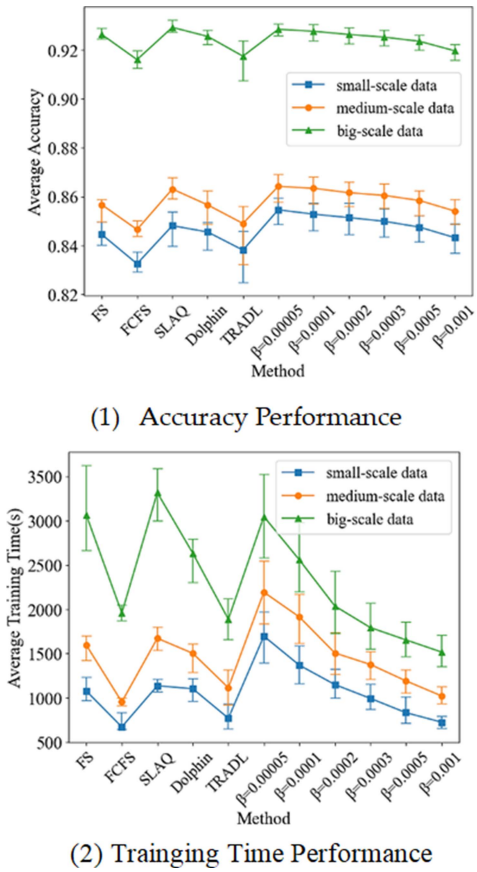

Fig. 7. Performance of different methods on different scales of data.

This avoids processing performance degradation from running too many tasks on a single computing node.

2) Resource Scheduling Experiment: **1. Performance Comparison**

For the resource scheduling experiment, several resource scheduling methods were selected as the baseline methods: (1) FS, in which the distributed learning task allocates all computing resources equally, (2) FCFS, in which, when a task is executed, the task occupies all computing resources, (3) SLAQ, (4) Dolphin, and (5) TRADL.

(1) Experiment 1

The experiment consists of 12 computing nodes divided into three compute clusters. The learning task involved only CNN-based image classification tasks, with a total of 12 computational tasks, using 3 types of data sets of different sizes. In order to objectively reflect the impact of β on accuracy and time, the same specified β value is used for different data sizes. Each method is repeated 20 times to achieve stable results.

Fig. 7 shows the performance of the models trained by different methods on different scales of data. "$\beta = 0.00005$" in the figure is a simplified representation of "RLPTO $\beta = 0.00005$". The proposed scheduling method relies on the reward function, which takes into account both accuracy and training time. At multiple β values, RLPTO can achieve higher model accuracy and shorter training time than other methods on small-scale data. The accuracy performance of RLPTO is comparable to SLAQ and higher than other methods on medium and large scale data. However, on large-scale data, the training time of RLPTO is shorter than that of FS, SLAQ and Dolphin in most cases. This shows that RLPTO tends to use more compute nodes to improve the training efficiency when facing learning tasks with large-scale data. Shorter training times come at the expense of some accuracy. If a user requires higher accuracy, the user can adjust the β value for better accuracy performance.

Table II shows the average accuracy of the model trained on three datasets of different sizes and the average total time for each method to complete the 12 tasks.

Using the FS method, the numbers of computing nodes allocated to the three parameter servers were (3, 3, 4). Because each computing node only computed a portion of the data set, the accuracy of the distributed learning task model and the task completion time were both at an intermediate level.

In the FCFS scheduling experiment, all compute nodes were used in the execution of each task, but the data amount of each compute node was the least, resulting in a large difference in calculated gradients, so the model accuracy was poor and the model convergence was slow, resulting in a longer overall training time.

TABLE II
MODEL PERFORMANCE AND EFFICIENCY WITH 12 COMPUTING NODES

Method	Computing nodes	Model average accuracy (%)	Average training time (s)
FS	4	87.59	8498
FCFS	10	86.51	14333
SLAQ	dynamic	88.07	9043
Dolphin	dynamic	87.61	7617
TRADL	dynamic	86.82	5388
RLPTO β=0.00005	dynamic	88.24	10835
RLPTO β=0.0001	**dynamic**	**88.13**	**8882**
RLPTO β=0.0002	dynamic	87.98	6898
RLPTO β=0.0003	dynamic	87.86	6069
RLPTO β=0.0005	dynamic	87.65	5303
RLPTO β=0.001	dynamic	87.23	4515

TABLE III
MODEL PERFORMANCE AND EFFICIENCY WITH 16 COMPUTING NODES

Method	Computing nodes	Model average accuracy (%)	Average training time (s)
FS	4	87.61	8139
FCFS	12	85.91	14682
SLAQ	dynamic	88.07	9043
Dolphin	dynamic	87.67	7663
TRADL	dynamic	86.94	5281
RLPTO 0.00005	dynamic	88.25	9469
RLPTO 0.0001	**dynamic**	**88.13**	**7602**
RLPTO 0.0002	dynamic	87.99	6479
RLPTO 0.0003	dynamic	87.88	5834
RLPTO 0.0005	dynamic	87.69	5334
RLPTO 0.001	dynamic	87.27	4597

SLAQ scheduling method takes model performance as evaluation criterion. Because the change value of loss function shrinks continuously in the process of distributed learning and training, the performance of the obtained model is higher than FCFS, Dolphin and TRADL, but the completion time is longer than FS, Dolphin, TRADL and RLPTO (β>0.00005).

The Dolphin algorithm pays more attention to the communication time and training time. The algorithm achieved better scheduling in terms of time, but the model accuracy of the distributed learning system was reduced.

TRADL is divided into two stages for training, when the accuracy of the model reaches a goal, the model is handed over to the user, so its training process is fastest.

The RLPTO resource scheduling method uses the dual indicators of model accuracy and training time to jointly judge the training stage of the tasks. As the β value increased, the average accuracy of the models decreased, and the task completion time also decreased. When β <0.0001, the average accuracy of RLPTO was higher than other methods, but the task completion time was relatively long, and the average accuracy of RLPTO was slightly lower when β>0.0005. When $\beta = 0.0001$, RLPTO enables the model to achieve higher accuracy and shorter training times than SLAQ.

(2) Experiment 2

In this study, another set of experiments was conducted to verify the scalability of the method, which included 16 compute nodes divided into 4 compute clusters, with the remaining conditions unchanged. The experimental results are shown in Table III, and the average accuracy of different methods is not much different from that of experiment 1. It is worth noting that this study uses Docker to build distributed machine learning clusters. As the number of compute nodes increases, the computing power of a single compute node decreases slightly, so the total task completion time does not decrease significantly.

2. The β value

Fig. 8 shows the effects of β on the model accuracy and training time for three different datasets of different sizes and 2 types of tasks. Due to differences in accuracy and training time under different scale data, the data shown in the figure is

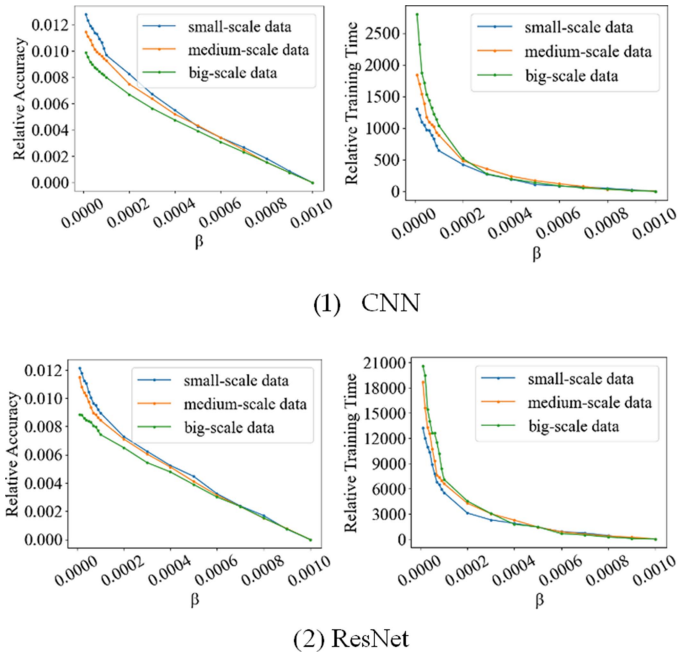

Fig. 8. Influence of β value on the accuracy and training time of CNN model under different data scales.

TABLE IV
RECOMMENDED β VALUES

Data scale	Recommended β value
small-scale data	[0.00005, 0.0002]
medium-scale data	[0.0001, 0.0002]
big-scale data	[0.0001, 0.0003]

the difference from the lowest value, which is called relative accuracy and relative training time. This allows for a better description of how accuracy and training time change with β value. As can be seen from the figure, as the β value increases, the accuracy of the model decreases and the time to complete the task decreases. The value of β can be determined by the user according to their own needs, if the user wants a higher accuracy, it is recommended that the value of β range is [0.00003, 0.0001], if the user wants a shorter training time, the recommended value of β range is [0.0003, 0.0005]. Considering the accuracy and training time, the reasonable range of β values is [0.0001, 0.0003]. It can be seen from the figure that the model accuracy trained on the large-scale data is less affected by the value of β, but the training time is more affected by the value of β. The accuracy of the model trained on small-scale data is greatly affected by the β value, but the training time is less affected by the β value. Therefore, this paper suggests that when the data size is small, choose a smaller β to pursue higher accuracy, and when the data size is large, choose a larger β to pursue a shorter training time. The recommended beta values are shown in Table IV.

3. Number of computing nodes

Fig. 9 shows the impact of the number of computing nodes on the accuracy and training time of a single learning task. It

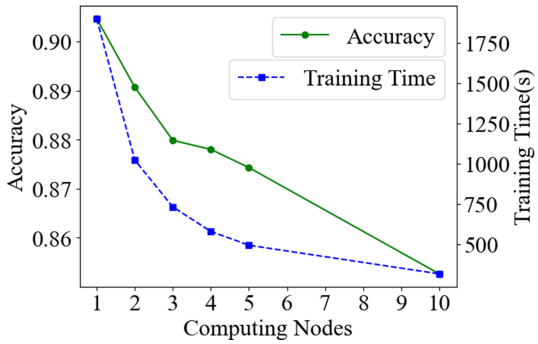

Fig. 9. Impact of the number of computing nodes on accuracy and training time.

TABLE V
TASK COMBINATIONS

No.	Priority: Normal 1, 2, 3	Priority: Highest 4	Priority: Urgent 5	priority sequence
1	12	0	0	1,2,2,3,1,2,1,3,1,1,1,3
2	8	4	0	2,3,1,4,1,2,2,1,4,1,1,3
3	6	6	0	1,4,2,3,1,4,4,3,4,4,1,4
4	6	4	2	1,4,4,3,1,2,5,3,4,4,5,1

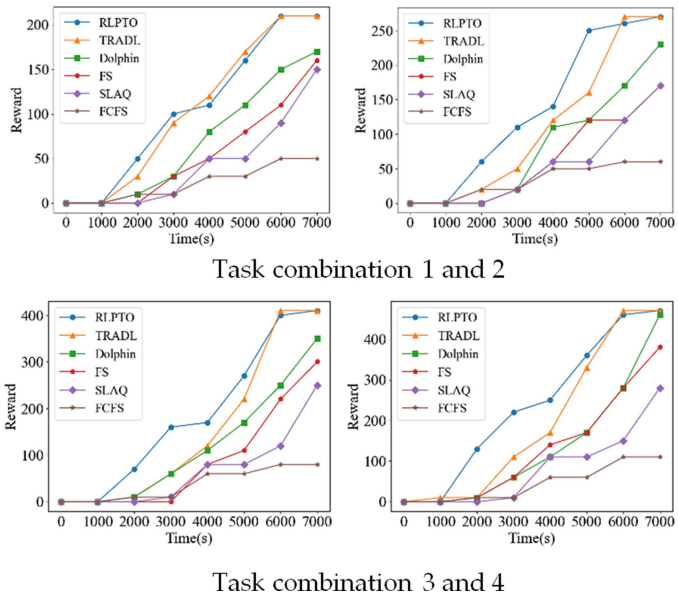

Fig. 10. Experimental results of task combinations with different priorities.

can be seen from the figure that as the number of computing nodes increases, the training time continuously decreases but the accuracy gradually decreases. This is because when the training data is divided into multiple parts and distributed to different computing nodes, each computing node can only use a portion of the data for training, which can lead to a decrease in the performance of the model on a single node, resulting in a decrease in the accuracy of the entire task. In addition, more computing nodes bring higher communication overhead, so the time efficiency benefits of increasing computing nodes will gradually decrease. Overall, when the number of computing nodes of a task is between 2 and 4, there is a more balanced performance in training time and accuracy.

4. Distributed Learning System Reward

The distributed learning system throughput is the total reward of the distributed learning tasks within a certain time. When multiple tasks have different urgency, the rewards for performing tasks with different urgency are different. In the experiment, it was supposed that the award of the highest-priority task was 80 points, and other tasks were given 10 to 40 points according to the task priority. There are a total of 12 tasks in this experiment, and the system randomly assigns priority to 12 tasks, which use three datasets of different sizes. The first four tasks used large-scale data sets, the middle four used medium-scale data sets, and the last four used small-scale data sets. The 12 tasks form several task combinations with different priority distributions, as shown in Table V.

The training time for this experiment is 120 minutes, and the reward score is recorded every 1000 seconds. In order to make the experimental comparison more reliable, we did not let β be 0.0001 in this experiment, but let β be a more general value of 0.0003. The TRADL algorithm has a short training time due to early termination, so the reward score will not increase after 100 minutes. The experimental results of different priority task combinations are shown in Fig. 10. The results indicate that RLPTO is beneficial for distributed learning clusters to schedule tasks and resources according to the needs of different users, respond to urgent or timely processing of high priority tasks. From the beginning to 60 minutes, there are sufficient tasks to be scheduled, and the RLPTO algorithm obtains more task rewards faster than other algorithms. The number of tasks in this experiment is fixed, but in reality, machine learning tasks are generated by users without time constraints. When there are many tasks that need to be scheduled, the RLPTO algorithm can respond to high priority tasks in a timely manner, which can maximize the overall system performance of the distributed learning cluster.

In fact, the start time of each learning task is different. Therefore, this paper takes the average reward of each resource scheduling method as the evaluation index. Fig. 11 shows the average reward of different resource scheduling algorithms under different task combinations. As can be seen from the figure, the proposed RLPTO method obtains the highest reward value under the four task combinations and shows very good learning task throughput.

VIII. DISCUSSION

When the reinforcement learning method is used to schedule distributed computing resources, each action is to adjust the resources of one parameter server. Therefore, the amount of resources adjusted by an action accounts for a small proportion of the whole cluster resources. As a result, the resource scheduling is not fast enough. In addition, the method mainly considers the

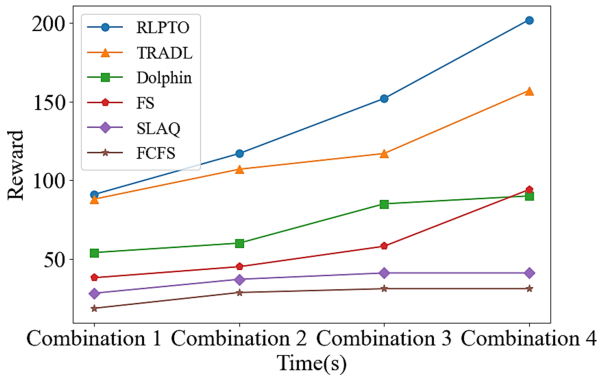

Fig. 11. Average reward of different task resource scheduling algorithms.

scheduling of computing resources, and does not consider the data management and scheduling in machine learning.

The scheduling algorithm proposed in this paper reduces the training time of machine learning tasks, so that limited compute nodes can handle more machine learning tasks per unit time. The algorithm improves the efficiency of resource utilization, so as to reduce the total number of computing nodes, and reduces the energy consumption.

IX. Conclusion

This paper proposed a task volume prediction method called LSP based on the combination of long- and short-term data, which can more accurately predict the future task volume. It then calculates the most appropriate number of working nodes via a heuristic algorithm. The results of experiments show that the resource allocation scheme based on LSP can relieve the task pressure of computing nodes and ensure the normal operation of nodes.

In terms of task scheduling, three task queues are established according to the type and priority of the task. Tasks are assigned to one of three task queues according to their urgency, ensuring that urgent and high-priority tasks can start training earlier. Moreover, a reinforcement learning-based resource scheduling strategy was proposed. Based on the model performance and training time, the reward function, action space, and state space of RLPTO were designed to dynamically adjust the resources occupied by every task. The results of experiments demonstrate that the model accuracy of large-scale data training is less affected by the value of β, but the training time is more affected by the value of β. The accuracy of the model trained on small-scale data is greatly affected by the β value, but the training time is less affected by the β value. RLPTO scheduling method tends to use more computing nodes when facing tasks with large scale data. This paper suppose to determine β value based on the data scale. Small-scale data can use smaller β to pursue higher accuracy. Large scale data should choose a larger β to reduce the training time.

In this paper, a large number of experiments are carried out on different scale data sets with different number of computing nodes. The experimental results show that the resource scheduling method RLPTO proposed in this paper has good scalability. RLPTO combines resource scheduling with task scheduling to maximize the reward of distributed computing clusters. Experiments show that compared with FS, FCFS, SLAQ, Dolphin and TRADL, RLPTO can make the computing cluster get the maximum reward in different task combinations. In the future, we will further study how to improve the scheduling efficiency of RLPTO.

References

[1] C.-L. Chang, H.-M. Tseng, and H.-T. Chu, "Fireworks image classification with deep learning," in *Proc. Int. Conf. Technol. Appl. Artif. Intell.*, 2021, pp. 311–314, doi: 10.1109/TAAI54685.2021.00067.

[2] Z. Huang, C. O. Dumitru, and J. Ren, "Physics-aware feature learning of SAR images with deep neural networks: A case study," in *Proc. IEEE Int. Geosci. Remote Sens. Symp.*, 2021, pp. 1264–1267, doi: 10.1109/IGARSS47720.2021.9554842.

[3] P. Wang, "Research and design of smart home speech recognition system based on deep learning," in *Proc. Int. Conf. Comput. Vis. Image Deep Learn.*, 2020, pp. 218–221, doi: 10.1109/CVIDL51233.2020.00-98.

[4] S. Wan, "Research on speech separation and recognition algorithm based on deep learning," in *Proc. IEEE Int. Conf. Power Intell. Comput. Syst.*, 2021, pp. 722–725, doi: 10.1109/ICPICS52425.2021.9524204.

[5] H. A. C. Okan Sakar, "A dynamic recurrent neural networks-based recommendation system for banking customers," in *Proc. 29th Signal Process. Commun. Appl. Conf.*, 2021, pp. 1–4, doi: 10.1109/SIU53274.2021.9477967.

[6] G. Huang, "E-commerce intelligent recommendation system based on deep learning," in *Proc. IEEE Asia-Pacific Conf. Image Process. Electron. Comput.*, 2022, pp. 1154–1157, doi: 10.1109/IPEC54454.2022.9777500.

[7] S. Roy et al., "Deep learning for classification and localization of COVID-19 markers in point-of-care lung ultrasound," *IEEE Trans. Med. Imag.*, vol. 39, no. 8, pp. 2676–2687, Aug. 2020, doi: 10.1109/TMI.2020.2994459.

[8] S. Xue et al., "Development of a deep learning method for CT-free correction for an ultra-long axial field of view PET scanner," in *Proc. IEEE 43rd Annu. Int. Conf. Eng. Med. Biol. Soc.*, 2021, pp. 4120–4122, doi: 10.1109/EMBC46164.2021.9630590.

[9] M. Patel, A. Das, V. K. Pant, and M. J., "Detection of tuberculosis in radiographs using deep learning-based ensemble methods," in *Proc. Smart Technol. Commun. Robot.*, 2021, pp. 1–7, doi: 10.1109/STCR51658.2021.9588936.

[10] S. Kim, N. Pham, W. Baek, and Y. Choi, "Machine-learning based performance estimation for distributed parallel applications in virtualized heterogeneous clusters," in *Proc. IEEE 37th Int. Conf. Distrib. Comput. Syst.*, 2017, pp. 2610–2611, doi: 10.1109/ICDCS.2017.310.

[11] W. Xiao et al., "Distributed graph computation meets machine learning," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 7, pp. 1588–1604, Jul. 2020, doi: 10.1109/TPDS.2020.2970047.

[12] A. Abd Elrahman, M. El Helw, R. Elshawi, and S. Sakr, "D-smartML: A distributed automated machine learning framework," in *Proc. IEEE 40th Int. Conf. Distrib. Comput. Syst.*, 2020, pp. 1215–1218, doi: 10.1109/ICDCS47774.2020.00115.

[13] M. Abadi et al., "Tensorflow: A system for large-scale machine learning," in *Proc.12th USENIX Symp. Operating Syst. Des. Implementation*, Savannah, 2016, pp. 265–283.

[14] D. Amodei et al., "Deep speech 2: End-to-end speech recognition in English and Mandarin," in *Proc. Int. Conf. Mach. Learn.*, New York, NY, USA City, 2016, pp. 173–182.

[15] F. N. Iandola et al., "FireCaffe: Near-linear acceleration of deep neural network training on compute clusters," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Las Vegas, 2016, pp. 2592–2600.

[16] S. Jeon et al., "MapReduce tuning to improve distributed machine learning performance," in *Proc. IEEE 1st Int. Conf. Artif. Intell. Knowl. Eng.*, 2018, pp. 198–200, doi: 10.1109/AIKE.2018.00045.

[17] W. Zhao and J. A. Stankovic, "Performance analysis of FCFS and improved FCFS scheduling algorithms for dynamic real-time computer systems," in *Proc. Real-Time Syst. Symp.*, 1989, pp. 156–165.

[18] M. Isard et al., "Quincy: Fair scheduling for distributed computing clusters," in *Proc. ACM SIGOPS 22nd Symp. Operating Syst. Princ.*, 2009, pp. 261–276.

[19] M. Zaharia et al., "Delay scheduling: A simple technique for achieving locality and fairness in cluster scheduling," in *Proc. 5th Eur. Conf. Comput. Syst.*, 2010, pp. 265–278.
[20] W. Wang, B. Li, and B. Liang, "Dominant resource fairness in cloud computing systems with heterogeneous servers," in *Proc. IEEE Conf. Comput. Commun.*, 2014, pp. 583–591.
[21] F. Yan et al., "Performance modeling and scalability optimization of distributed deep learning systems," in *Proc. 21th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, Sydney, 2015, pp. 1355–1364.
[22] Y. S. L. Lee et al., "Dolphin: Runtime optimization for distributed machine learning," in *Proc. Int. Conf. Mach. Learn. ML Syst. Workshop*, New York, NY, USA City, 2016, pp. 1–14.
[23] C. Gao, R. Ren, and H. Cai, "GAI: A centralized tree-based scheduler for machine learning workload in large shared clusters," in *Proc. Int. Conf. Algorithms Architectures Parallel Process.*, Cham, Switzerland: Springer, 2018, pp. 611–629.
[24] J. Gu et al., "Tiresias: A GPU cluster manager for distributed deep learning," in *Proc. 16th USENIX Symp. Networked Syst. Des. Implementation*, 2019, pp. 485–500.
[25] Y. Peng et al., "Optimus: An efficient dynamic resource scheduler for deep learning clusters," in *Proce. 13th EuroSys Conf.*, 2018, pp. 1–14.
[26] H. Zhang et al., "SLAQ: Quality-driven scheduling for distributed machine learning," in *Proc. Symp. Cloud Comput.*, 2017, pp. 390–404.
[27] W. Zheng et al., "Target-based resource allocation for deep learning applications in a multi-tenancy system," in *Proc. IEEE High Perform. Extreme Comput. Conf.*, 2019, pp. 1–7.
[28] Y. S. L. Lee et al., "Dolphin: Runtime optimization for distributed machine learning," in *Proc. ICML ML Syst. Workshop*, 2016. pp. 1–14.
[29] R. Hyndman, "Time series data library [EB/OL]," 2018. [Online]. Available: https://pkg.yangzhuoranyang.com/tsdl

Xiaofeng Lu (Member, IEEE) received the PhD degree from the Beijing University of Aeronautics & Astronautics, Beijing, China, in 2010. He is an associate professor with the School of Cyberspace Security, Beijing University of Post and Telecommunications. During his PhD, he held visiting scholar positions with the Computer Laboratory, University of Cambridge, U.K. His main research interests include cyberspace security, information security, and artificial intelligence.

Chao Liu received the MS degree in cyberspace security from the Beijing University of Posts and Telecommunications, China in 2022. His current research interests include distributed machine learning and resource scheduling.

Senhao Zhu received the BS degree in computer science and technology from the Beijing University of Posts and Telecommunications, in 2023. He is currently working toward the MS degree with the School of Cyberspace Security, Beijing University of Posts and Telecommunications, China. His current research interests include distributed machine learning and resource scheduling.

Yilu Mao received the BS and PhD degrees in computer science and technology from the China University of Science and Technology, China, in 2004 and 2009. He is the director with Alibaba Cloud Intelligence Business Group. His current research interests include cloud computing, OS, and resource scheduling.

Pietro Lio is a professor with the University of Cambridge, Computer Laboratory in England and fellow and director of Studies with the Fitzwilliam College of University of Cambridge. He is currently modeling biological processes on networks; modeling stem cells; developing transcription and phylogenetic applications on a grid environment. He is also interested in bio-inspired design of wireless networks, epidemiological networks

Pan Hui (Fellow, IEEE) is a chair professor of computational media and arts and director of the Centre for Metaverse Research with the Hong Kong University of Science and Technology (Guangzhou) and a chair professor of Emerging Interdisciplinary Areas, Hong Kong University of Science and Technology. He is also the Nokia chair in data science with the University of Helsinki. He is a leading expert in Augmented Reality and Mobile Computing, with more than 450 research papers, 32 patents, and more than 27,000 citations. He is an International fellow of the Royal Academy of Engineering, an ACM distinguished scientist, and a member of the Academia Europaea.