RLSK: A Job Scheduler for Federated Kubernetes Clusters based on Reinforcement Learning

Jiaming Huang, Chuming Xiao, Weigang Wu
School of Data and Computer Science, Sun Yat-sen University, China
{*huangjm39, xiaochm*}*@mail2.sysu.edu.cn*
wuweig@mail.sysu.edu.cn

***Abstract*—Job scheduling in cluster is often considered as a difficult online decision-making problem, and its solution depends largely on the understanding of the workload and environment. People usually first propose a simple heuristic scheduling algorithm, and then perform repeated and tedious manual tests and adjustments based on the characteristics of the workload to gradually improve the algorithm. In this work, focusing on multi-cluster environments, load balancing and efficient scheduling, we present RLSK, a deep reinforcement learning based job scheduler for scheduling independent batch jobs among multiple federated cloud computing clusters adaptively. By directly specifying high-level scheduling targets, RLSK interacts with the system environment and automatically learns scheduling strategies from experience without any prior knowledge assumed over the underlying multi-cluster environment and human instructions, which avoids people's tedious testing and tuning work. We implement our scheduler based on Kubernetes, and conduct simulations to evaluate the performance of our design. The results show that, RLSK can outperform traditional scheduling algorithms.**

***Index Terms*—Deep reinforcement learning, kubernetes, job scheduling, resource management**

I. Introduction

With the development of cloud computing, the dominating virtualization technology for data centers has gradually changed from virtual machine to container. Among others, kubernetes, an open-source container cluster management system developed by Google, has become a popular platform for large-scale container-based cloud computing clusters.

Although cloud data centers are now running more and more diverse tasks, batch jobs still account for a large proportion [1]. In a cloud data center, batch jobs are submitted to a centralized job scheduler, which picks jobs to run once the resources required are available. In most cases, the scheduling problem is NP-hard or NP-complete.

In the traditional resource management problems, a lot of sophisticated heuristics have been designed, such as fair scheduling [2] [3], first-fit [4], simple packing strategies [5]. These heuristics prioritize generality, ease of understanding, and straightforward implementation over achieving the ideal performance on a specific workload. In addition, many complex and tedious meta-heuristic algorithms have been proposed for specific loads, such as scheduling with genetic algorithm [6] or ant colony algorithm [7]. In most cases, people prefer to propose a simple heuristic scheduling algorithm for specific scenarios first, and then carry out repeated and tedious manual tests and adjustments to gradually improve the algorithm, so that it can make more effective use of the overall resources of the system, while ensuring fairness and scheduling efficiency. These handcrafted calculation methods are general for the workload but the adjustment of the parameters relies on the operator's experience. Due to the complexity of the relationship between parameters and optimization goals, the process is cumbersome and the improvement cannot be guaranteed.

In recent years, reinforcement learning has been used to handle decision-making problems especially job scheduling in cloud data centers. DeepRM [8] uses deep reinforcement learning to schedule jobs by representing the states as images. Decima [9] solves the scheduling problem of dependent tasks in a Spark cluster by combining deep reinforcement learning and graph convolutional network. W. Chen [20] improves DeepRM [8] and realizes the management of resources in a multi-cluster and multi-resource environment, but with the same state representation with DeepRM [8]. A. Orhean [22] uses reinforcement learning to schedule tasks with dependencies on multiple hosts. The scheduling goals of the above works are reducing the makespan and slowdown. N. Liu proposes a hierarchical architecture [21], using deep reinforcement learning technology to solve the resource allocation and power consumption management problems in a single cluster. Furthermore, reinforcement learning has been applied in virtual machine configuration [10] and making virtual machine online migration decisions [11].

In traditional reinforcement learning, we need to quantify the system state and control actions first, and then establish a mapping tabular, which is limited by dimensions. When it comes to job scheduling in cluster, the dimensions of state and action spaces will increase exponentially with the number of machines and resources, thus greatly hindering the application of reinforcement learning in complex cluster environments. For example, a state in the state space may be the Cartesian product of characteristics of current resource utilization of each server as well as current jobs. What's more, with the widespread of multi-cluster and hybrid clouds, job scheduling inevitably becomes more difficult, since different resources of multiple clusters need to be considered simulteneously. Without proper handling, the state and action space of reinforcement learning may be too large, which will have a negative impact on learning. Therefore, after carefully shaping the reward, the way to represent the state and action spaces, make them map to each other, and handle the logic

978-1-7281-1099-8/20/$31.00 ©2020 IEEE
DOI 10.1109/IC2E48712.2020.00019

of scheduling within and between clusters is the key to applying reinforcement learning to a multi-cluster scheduling environment.

In this paper, considering a federated kubernetes clusters environment, we designed RLSK, a multi-cluster scheduler based on deep reinforcement learning, then implemented the scheduler on the kubernetes platform, finally we performed a simulation to measure the effect. RLSK can directly learn how to schedule jobs among multiple clusters based on history to balance the average utilization of resources among clusters and make the utilization of different resources within each cluster tend to balance, so that it can improve the total utilization of resources.

The rest of the paper is organized as follows. We present background knowledge in Section II, including basic reinforce learning technique and the kubernetes platform. The details of RLSK are presented in Section III, and performance evaluation is reported in Section IV. Finally, Section V concludes the paper.

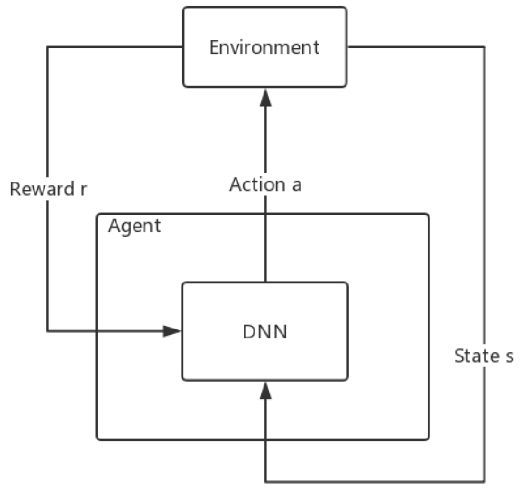

Fig. 1. Reinforcement Learning with DQN

II. Background

We briefly introduce the technique of reinforcement learning and kubernetes.

A. Reinforcement Learning

Agent-Environment Interaction. As illustrated in Fig. 1, RL [12] is a process of learning that generates an optimal action policy on a given environment state through interacting with a dynamic environment. At each time step t, the agent observes some state s_t, and chooses an action a_t. Following the action, the state of the environment transitions to s_{t+1} and the agent receives reward r_t. The state transitions and rewards are stochastic and assumed to have the Markov [13] property. That is, at each step, the state transition probabilities and rewards depend only on the state of the environment and the action taken by the agent. The goal of learning is to maximize the expected cumulative discounted reward: $\mathbb{E}[\sum_{t=0}^{\infty} \gamma_t r_t]$, where $\gamma \in (0, 1]$ is a factor discounting future rewards.

Algorithm 1: deep Q-learning with experience replay.

Initialize replay memory D to capacity N;
Initialize action-value function Q with random weights θ;
Initialize target action-value function $\hat{Q}$ with weights $\theta^- = \theta$;
for *episode=1,M* **do**
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$;
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \arg\max_a Q(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D
    Sample random minibatch oftransitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D
    Set $y_j =$
    $\begin{cases} r_j, \text{if episode terminates at step } j+1 \\ r_j + \gamma \max_d \hat{Q}(\phi_{j+1}, a'; \theta^-), \text{otherwise} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to the network parameters θ
    Every C steps reset $\hat{Q} = Q$
  end
end

Q-Learning and DQN. Q-learning [14] is a value based training method proposed by Watkins in traditional reinforcement learning. The Q-learning consists of an agent, a set of states S of the world, a set of actions A, a definition of how actions change the world $T : S \times A \to S$, also known as transition dynamics, a set of rewards $R : S \times A \to R$ for each actions, a table of utilities $Q : S \times A \to R$ and a policy $\pi : S \to A$. The goal of the agent is to maximize the reward. in order to do so, it must learn which is the best action taken from each state, i.e., the optimal action having the highest long-term reward. For such a solution to be effective, an agent should run multiple training episodes for the purpose of exploring and finding the optimal policy.

And the value updating rule is given by Formula (1):

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)] \quad (1)$$

Deep Q-learning combines Q-learning with deep neural network, which is proposed by Google DeepMind and used in playing the Atari games [15], [16]. DQN uses a neural network as the function fitting to replace Q table to generate Q value, so as to solve the problem of storage and representation when the dimension of the state or action is high and continuous. The deep neural network has a good effect on the extraction

of complex features. The process of nature DQN is shown in Algorithm 1. In our design, nature DQN is used to train the agent, which converges quickly and produces good scheduling results.

B. Kubernetes

Kubernetes [17] is developed by Google based on Borg [18]. Because kubernetes has a good scheduling ability for container-based environment and is convenient for expansion and contraction management, it is very friendly for application development in the cloud environment. Due to such advantages, kubernetes has become more and more popular in past years, and adopted by many industry entities.

The main architecture of kubernetes is shown in Fig. 2. Kubernetes adopts the master-slave mode, and the scheduler kube-scheduler is used to bind the pod to the most appropriate work node according to various scheduling algorithms. The whole scheduling process is divided into two stages: the pre-selection strategy and the priority strategy.

Predicates: the kube-scheduler filters out nodes that do not meet the policy according to the pre-selection policy. For example, the resources of a node are not enough to meet the needs of the job.

Priorities: according to the priority strategy, the priority will rank the pre-selected nodes and select the node with the highest score for task allocation. For example, nodes with richer resources and less load may have higher ranking.

In addition, the management of kubernetes multiple clusters is realized by Federation. Federation schedules different pods to different kubernetes clusters based on the needs of users. Generally, it schedules the workload to different kubernetes clusters according to the requirements of application geographic areas. For different end users, it provides higher bandwidth and lower latency.

This is different from the goal of multi cluster scheduler we designed. From the perspective of managers or cloud providers, we train the scheduler to adaptively balance the load between different clusters, and reduce the utilization difference of different resources within the cluster.

III. DESIGN

In this section, we present the design of RLSK, including the architecture of online multi-cluster multi-resource scheduling with RL and the implementation of job scheduling module.

A. Job Scheduling Scenario

For simplicity, we consider that there are a number m of homogeneous clusters, and each cluster has the same type and quantity of resources. Each cluster has totally D kinds of resources (for example, CPU and memory, etc.), and jobs are continuously arrived according under the online mode.

At each scheduling point, RLSK can schedule the current job to the most appropriate cluster through the automatically learned strategy. Similar to the hypothesis in existing research [19], we assume that the resource requirements of each task are known when they arrive at RLSK, and the resource request for the job can be expressed as a vector, e.g., $S_J = (r_{j1}, \ldots, r_{jk}, \ldots, r_{jd})$, where j is the job id, and r_{jk} represents a certain resource requirement of job j.

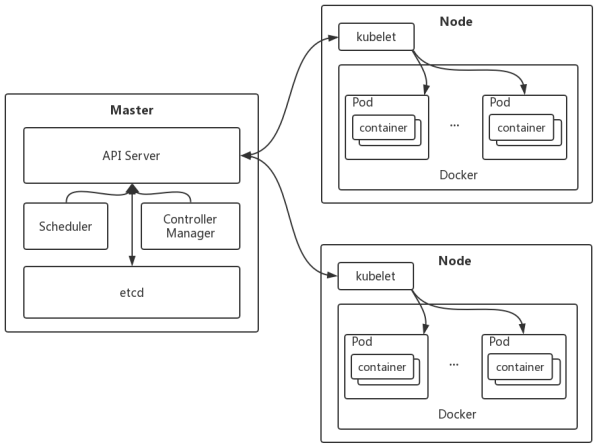

Fig. 2. Architecture diagram of kubernetes

In addition, we assume that there is no preemption and the requested resources will not change. That is, once a task is scheduled successfully, the resources allocated to it will not be recycled by the scheduler until the task is completed. Moreover, we regard each cluster as a resource pool, i.e., we do not consider resource fragmentation within the cluster.

In addition to ensuring the overall utilization of each cluster, we have two basic scheduling objectives. One is to achieve the balance of average resource utilization among different clusters, and the other is to minimize the utilization gap between different resources for each cluster.

B. The Framework of RLSK

The overall architecture of RLSK is shown in Fig. 3. RLSK adopts a centralized scheduling mode to schedule jobs by interacting with master nodes in each kubernetes cluster.

As RLSK runs outside of each cluster, in order to collect the data of each kubernetes cluster and deliver jobs to the corresponding cluster, an assistant kubernetes module needs to be deployed on the master node of each kubernetes cluster, which is responsible for the information communication between the RLSK scheduler and kubernetes master. It is deployed at the master node of each kubernetes cluster. After accessing the cluster, it can perform some operations on the cluster by the kubernetes native API.

RLSK receives the batch job information submitted by users, and then combines the currently collected status information of each cluster from the assistant module, makes the scheduling decision with the trained agent inside. Finally, it delivers the batch job to the corresponding cluster.

The main functions of the assistant module are described as follows.

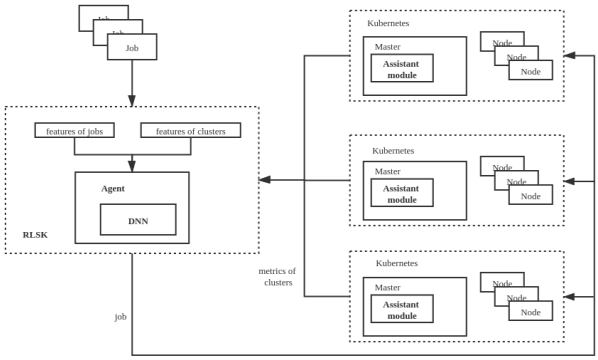

Fig. 3. Overall architecture of the proposed RLSK scheduler

1) Collecting cluster data. By calling the API provided by kubernetes, RLSK can traverse each node in the cluster, the pod running at the node and the container in the pod, so as to collect resource state information (e.g., CPU and memory utilization) of the cluster.
2) Receiving jobs from the scheduler, and delivering them to kubernetes internal scheduler. In order to simplify the presentation, the execution of each job is limited to create a pod resource object containing only one container independently.
3) Managing the pods, containers and other resource objects, at all nodes in the cluster.

C. Model design

To solve the multi-cluster multi-resource scheduling problem, we design a new model by delicately designing the states, actions, and rewards of the reinforcement learning model.

*1) **States**:* We express the environment state of the system as a one-dimensional vector, which includes the current resource usage of each cluster and the features of jobs waiting to be scheduled. The resource usage of a cluster is represented by the occupancy rate of various resources in the cluster. In order to facilitate learning, the characteristics of the job are transformed into the ratios of the various resource requirements and the corresponding resources of the cluster. Therefore, the value range of each item of the state vector is $[0, 1]$. At this time, the environment state is defined as S^t. S^t is the collection of the overall state S_c^t of each cluster and state S_j of task j at time t, that is, $S^t = [S_c^t, S_j]$. For the overall state of cluster S_c^t, considering a total of M clusters, S_c^t consists of the state of each cluster, e.g. C_1^t, C_m^t. For the state C_i^t of a single cluster, it is composed of the occupancy of different resources within it. For job j, the requested utilization of a resource p is defined as U_{jp}, and the occupancy rate of a resource p of cluster C_i at time t is defined as U_{ip}^t.

Therefore, for the case of M clusters and D resources, the definition of environment state at time t can be defined as Formula (2).

$$\begin{aligned} S^t &= [S_c^t, S_j] \\ &= [C_1^t, C_2^t, \cdots, C_M^t, S_j] \\ &= [U_{11}^t, \cdots, U_{1D}^t, \cdots, U_{MD}^t, U_{j1}, \cdots, U_{jD}] \end{aligned} \tag{2}$$

*2) **Actions**:* Some of the settings in our model are similar to those in DeepRM [8]. For example, the time is discrete and jobs arrive discretely in the system. Agents are only activated to make decisions when jobs arrive. The scheduling action of the agent can be expressed by a finite number of discrete numbers. For example, $a = i$ means that the job should be scheduled to i^{th} cluster. In addition, considering that the agent can also decide to postpone processing by judging the current resource utilization of each cluster, we set a small strategy here. If the agent recognizes that the current system state is not suitable for scheduling this job (e.g., all clusters are currently in a high-load state), we will set the job to be randomly postponed for a certain time interval. Therefore, we use the number 0 to represent the action to delay the job for a random time interval, instead of scheduling the job at the current moment. Therefore, whenever the agent performs a scheduling action, a job is assigned to a certain cluster, or it is postponed for the scheduling in the next time step.

Then, if there are M clusters, the action space is defined as Formula (3).

$$A = \{a | a \in \{0, 1, 2, \cdots, M\}\} \tag{3}$$

*3) **Rewards**:* Our scheduling goal is not only to improve the overall resource utilization of multiple clusters, but also to balance the average utilization of each cluster, and to minimize the different resource utilization differences within the cluster, so as to avoid the scheduling bottleneck caused by single resource shortage.

However, there are some contradictions between these goals. For example, when new tasks are assigned to a cluster, the overall resource utilization will be improved but the load balance between clusters may be broken. Moreover, it may cause a gap in the utilization of different resources within the selected cluster. Therefore, a trade-off between the relative importance of these three goals is necessary. We can set the reward r as a linear combination of scalars corresponding to them. For example, reward can be equal to the overall utilization of the system at a certain time minus the degree of resource imbalance between or within clusters.

We define the reward function $r(t)$ as Equation (4).

$$r(t) = \alpha Util(t) - \beta DiffCluster(t) - \gamma DiffRes(t) \tag{4}$$

Obviously, the three weights here determine the effect of the three scheduling objectives. If we keep α and β unchanged and increase the value of γ, each cluster pay more attention to maintaining the balance of resources, which may cause jobs to be continuously delayed rather than scheduled as soon as possible. Therefore, parameter settings will significantly affect the overall scheduling efficiency and resource utilization.

As aforementioned, we mark M as the number of clusters and D as the number of resource types.

The resource utilization of a cluster is represented by the average value of the current internal resource utilization, in which the current utilization of a cluster m is recorded as shown in (5).

$$U_m(t) = \frac{\sum_{i=1}^{D} U_m^i}{D} \tag{5}$$

The overall resource utilization of each cluster is shown as Equation (6).

$$Util(t) = \sum_{m=1}^{M} U_m(t) \tag{6}$$

We have tried to use variance to express the utilization difference between different resources in a single cluster, but the effect is poor. Therefore, we directly calculate the absolute difference between the utilization of different resources, and then calculate their sum. This is intuitive but indeed effective. The overall balance of resource utilization in the cluster is expressed as Formula (7).

$$DiffCluster(t) = \sum_{m=1}^{M} \sum_{i=1}^{D} \sum_{j=i}^{D} |U_m^i - U_m^j| \tag{7}$$

Similarly, the utilization balance between clusters is expressed as (8).

$$DiffRes(t) = \sum_{i=1}^{M} \sum_{j=i}^{M} |U_i(t) - U_j(t)| \tag{8}$$

Algorithm 2: Training Algorithm

```
job_sequences = generate_job_sequence()
Initialize eval network Q, target network Q̂
for each iteration do
    for each execution sequence do
        for at each decision t_k do
            With probability ε select a random action,
              otherwise a_k = arg max_a Q(s_k, a), in
              which Q(s_k, a) is eval network;
            Perform system control using the chosen
              action;
            Observe state transition at next decision
              t_{k+1} with new state s_{k+1}, receive reward
              r_k(s_k, a_k) during time period [t_k, t_{k+1});
            Store transition (s_k, a_k, r_k, s_{k+1}) in D;
            Updating Q(s_k, a_k) based on r_k(s_k, a_k)
              and max_{a'} Q̂(s_{k+1}, a') with Q-learning
              updating rule in (1) and target network Q̂;
        end
        Update eval network Q using target network Q̂
          every certain number of steps;
    end
end
```

D. Training Algorithm

We use a deep feedforward neural network to calculate the Q value. It accepts the input of one-dimensional state vectors and outputs one-dimensional action vectors which represent the value of all possible actions.

To train an agent, we consider multiple examples of job arrival sequences and patterns. In each iteration, a set of fixed number of tasks will arrive online and be scheduled based on the agent. The state, action, reward and next state information for all scheduling decisions of each episode are recorded, and use these samples to update DNN.

At each decision t_k, the system is in state s_k. RLSK agent deduces $Q(s_k, a)$ of all possible actions with DNN, then uses $\epsilon - greedy$ strategy to select the largest $Q(s_k, a)$ with probability $1 - \epsilon$, or selects other random actions with probability ϵ. The selected action is recorded as a_k. At the next decision time t_{k+1}, RLSK agent performs a Q-value update based on the reward $r_k(s_k, a_k)$ in time $[t_k, t_{k+1}]$.

Only when a job sequence is successfully scheduled, will it be sequentially trained with the next job sequence. Algorithm 2 shows the pseudo-code of our training algorithm.

IV. Evaluation

To examine and analyze the performance of RLSK, we conduct experimental simulations. In the following, we firstly introduce the design of simulation, and then present the evaluation results. To show the advantages of RLSK, we compared it with existing scheduling algorithms designed for similar environments.

A. Setup

In this simulation experiment environment, we set up three homogeneous clusters. We assume two resources, with capacity 1r, 1r respectively. And we generate the workload similar to DeepRM [8]. Jobs arrive online according to a Bernoulli process. The duration and resource request are chosen as follows: 80% of the duration for jobs are uniformly chosen between 1t and 60t while the remaining are chosen uniformly from 200t to 300t. Furthermore, each job has a dominant resource which is choosen independently and randomly. The request for the dominant resource is chosen uniformly between 0.025r and 0.05r while the request of the other resource is chosen uniformly between 0.005r and 0.01r.

The four algorithms compared are First Fit (The scheduler will schedule the job to the first cluster that meets all requirements encountered in the traversal), Random (a method that schedules jobs to a random cluster), Round-Robin (a method that sequentially assigns jobs to different clusters in a polling manner) and Least Load (an algorithm that finds the cluster with the smallest load each time, and then schedules tasks to it).

The performance metrics measured in the evaluation are as follows,

1) the intuitive real-time resource utilization curves within the cluster.
2) the average resource utilization within each cluster, i.e. $mean = (cpu + memory)/2$.

3) the degree of utilization difference between different resources within the cluster, i.e. $(\frac{cpu}{mean}-1)^2+(\frac{memory}{mean}-1)^2$.
4) makespan, which is the maximum completion time in each cluster.

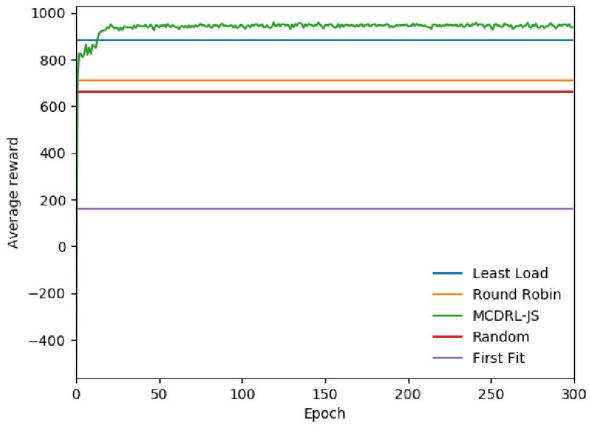

Fig. 4. Average rewards

B. Performance

Fig. 4 presents the detailed average reward obtained with our training process, which show that the agent can converge remarkably faster at the early stage. Please note that, although the reward received by RLSK at the beginning is negative, it starts to catch up with other algorithms after only a few dozens of epochs. And after convergence, the reward obtained by RLSK is significantly higher than other algorithms.

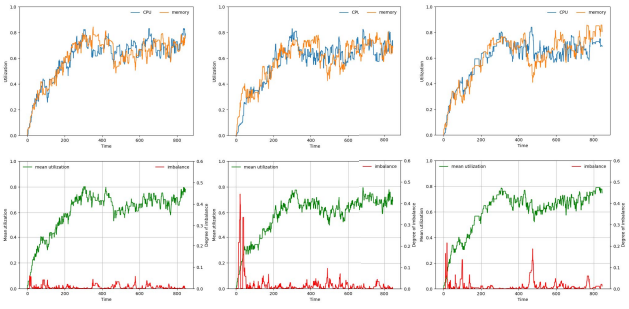

Fig. 5. RLSK Result

Fig. 5 to Fig. 9 show the effectiveness of scheduling of RLSK, Least Load, Round Robin, Random, and First Fit respectively. The comparison is conducted under the overall system load at about 70%. The first row of figures represents the internal utilization of CPU and memory for the three clusters in the scheduling process, and the second row represents the average utilization of the the clusters during the scheduling process. The fluctuations reflect the degree of utilization gap between the two resources within the three clusters.

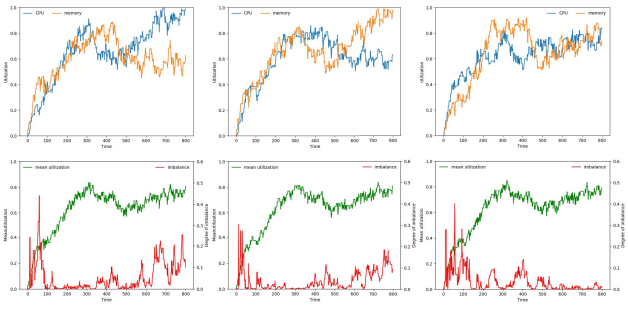

Fig. 6. Least Load Result

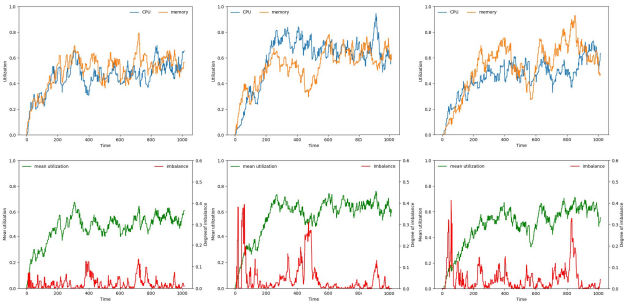

Fig. 7. Round Robin Result

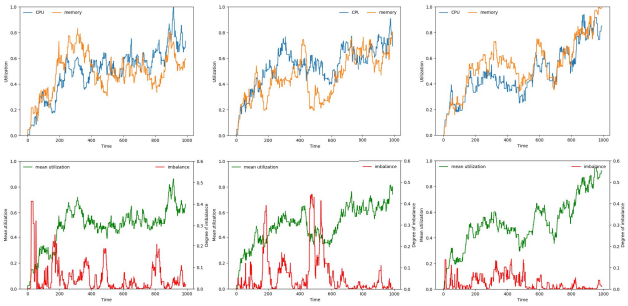

Fig. 8. Random Result

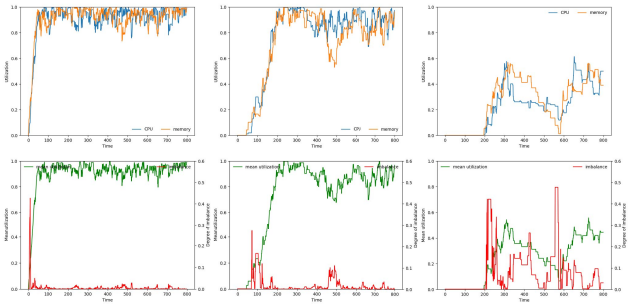

Fig. 9. First Fit Result

During the scheduling of RLSK, the average utilization of each cluster is quite stable and close to the average value, i.e., about 70% during the scheduling process.

As for the least load algorithm, it selects the cluster with the lowest resource utilization each time for the task. The utilization between different clusters can keep balanced, but the resources within each cluster cannot be as balanced as RLSK.

Although round robin maintains scheduling fairness, the balance of resources between different clusters cannot be achieved. Furthermore, there is also a large difference in the degree of resource utilization within the cluster at some times. Random scheduling can slightly maintain the fairness of scheduling, but it cannot maintain the balance in resource utilization due to its large blindness. The random scheduling has the same probability of selecting each cluster as a whole, but its blindness can easily cause load imbalance, and the difference in resources within the cluster is not considered. The effect of balancing resources is even worse than round robin. The results of first fit show that it cannot keep load balanced and not suitable for the scheduling scenario of multiple clusters.

TABLE I
MAKESPAN

Algorithm	RLSK	Least Load	Round Robin	Random	First Fit
makespan	1115.5	1081.3	1288.5	1263.8	1086.0

The average makespan of the algorithms is shown in Table 1. Some untrained job sequences are randomly selected, and different algorithms are tested. In order to achieve better load balancing, RLSK delays some jobs appropriately according to the current load balancing situation. Therefore, the makespan of RLSK is slightly higher than the makespan of Least-Load and First-Fit, but it outperforms Round-Robin and the Random algorithm. Considering that RLSK can improve load balancing and utilization significantly, the slightly larger makepan is acceptable.

V. Conclusion And Future Work

Based on kubernetes and deep reinforcement learning, we design RLSK, a multi-cluster job scheduler that can achieve resource balance both inside and outside a cluster. The adaptive scheduling of jobs among clusters are be realized by the deep reinforcement learning model, which makes the average utilization of resources among clusters and the utilization of different resources within each cluster tend to balance and improve the utilization of resources. In future, we will improve and extend our work in several directions, including more suitable neural network model to consider more information within the cluster, scheduling dependent jobs between multiple clusters, and scheduling the co-located jobs in a large scale cluster.

Acknowledgment

This work is partially supported by National Natural Science Foundation of China (U1711263, U1801266, U1811461), and Guangdong Natural Science Foundation (2018B030312002).

References

[1] Feitelson, D.G., Rudolph, L., Schwiegelshohn, U., Sevcik, K.C. and Wong, P., "Theory and Practice in Parallel Job Scheduling," in Workshop on Job Scheduling Strategies for Parallel Processing, 1997, pp. 1–34.

[2] Hadoop: Fair Scheduler. (n.d.). Retrieved April 12, 2019, from https://hadoop.apache.org/docs/current/hadoopyarn/hadoop-yarn-site/FairScheduler.html

[3] Ghodsi, A., Zaharia, M., Hindman, B., Konwinski, A., Shenker, S., and Stoica, I. 2011. Dominant Resource Fairness: Fair Allocation of Multiple Resource Types. In Nsdi (Vol. 11, No. 2011, pp. 24-24).

[4] Song, W., Xiao, Z., Chen, Q. and Luo, H., 2013. Adaptive resource provisioning for the cloud using online bin packing. IEEE Transactions on Computers, 63(11), pp.2647-2660.

[5] Grandl, R., Ananthanarayanan, G., Kandula, S., Rao, S. and Akella, A., 2014. Multi-resource packing for cluster schedulers. ACM SIGCOMM Computer Communication Review, 44(4), pp.455-466.

[6] Pezzella, F., Morganti, G. and Ciaschetti, G., 2008. A genetic algorithm for the flexible job-shop scheduling problem. Computers Operations Research, 35(10), pp.3202-3212.

[7] Azad, P. and Navimipour, N.J., 2017. An energy-aware task scheduling in the cloud computing using a hybrid cultural and ant colony optimization algorithm. International Journal of Cloud Applications and Computing (IJCAC), 7(4), pp.20-40.

[8] Mao, H., Alizadeh, M., Menache, I. and Kandula, S., 2016, November. Resource management with deep reinforcement learning. In Proceedings of the 15th ACM Workshop on Hot Topics in Networks (pp. 50-56). ACM.

[9] Mao, H., Schwarzkopf, M., Venkatakrishnan, S.B., Meng, Z. and Alizadeh, M., 2019. Learning scheduling algorithms for data processing clusters. In Proceedings of the ACM Special Interest Group on Data Communication (pp. 270-288).

[10] Yazdanov, L. and Fetzer, C., 2013, June. Vscaler: Autonomic virtual machine scaling. In 2013 IEEE Sixth International Conference on Cloud Computing (pp. 212-219). IEEE.

[11] Basu, D., Wang, X., Hong, Y., Chen, H. and Bressan, S., 2019. Learn-as-you-go with megh: Efficient live migration of virtual machines. IEEE Transactions on Parallel and Distributed Systems.

[12] Kaelbling, L.P., Littman, M.L. and Moore, A.W., 1996. Reinforcement learning: A survey. Journal of artificial intelligence research, 4, pp.237-285.

[13] Sigaud, O. and Buffet, O. eds., 2013. Markov decision processes in artificial intelligence. John Wiley Sons.

[14] Watkins, C.J. and Dayan, P., 1992. Q-learning. Machine learning, 8(3-4), pp.279-292.

[15] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G. and Petersen, S., 2015. Human-level control through deep reinforcement learning. Nature, 518(7540), p.529.

[16] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D. and Riedmiller, M., 2013. Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602.

[17] Burns, B., Grant, B., Oppenheimer, D., Brewer, E. and Wilkes, J., "Borg, omega, and kubernetes," ACM Queue, vol. 14, pp. 70–93, 2016.

[18] Verma, A., Pedrosa, L., Korupolu, M., Oppenheimer, D., Tune, E. and Wilkes, J., 2015, April. Large-scale cluster management at Google with Borg. In Proceedings of the Tenth European Conference on Computer Systems (p. 18). ACM.

[19] Grandl, R., Ananthanarayanan, G., Kandula, S., Rao, S. and Akella, A., 2014. Multi-resource packing for cluster schedulers. ACM SIGCOMM Computer Communication Review, 44(4), pp.455-466.

[20] Chen, W., Xu, Y. and Wu, X., 2017. Deep reinforcement learning for multi-resource multi-machine job scheduling. arXiv preprint arXiv:1711.07440.

[21] Liu, N., Li, Z., Xu, J., Xu, Z., Lin, S., Qiu, Q., Tang, J. and Wang, Y., 2017, June. A hierarchical framework of cloud resource allocation and power management using deep reinforcement learning. In 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS) (pp. 372-382). IEEE.

[22] Orhean, A.I., Pop, F. and Raicu, I., 2018. New scheduling approach using reinforcement learning for heterogeneous distributed systems. Journal of Parallel and Distributed Computing, 117, pp.292-302.