Deep Reinforcement Learning for Job Scheduling on Cluster

Zhenjie Yao[1,2,3](✉), Lan Chen[1,2], and He Zhang[1,2]

[1] Institute of Microelectronics, Chinese Academy of Sciences, Beijing, China
{yaozhenjie,chenlan,zhanghe}@ime.ac.cn

[2] Beijing Key Laboratory of Three-dimensional and Nanometer Integrated Circuit Design Automation Technology, Beijing, China

[3] Purple Mountain Laboratory: Networking, Communications and Security, Nanjing, China

Abstract. Job scheduling is a key function of cluster computing. Efficient job scheduling can improve hardware resource utilization and promote the execution efficiency of jobs. Conventional scheduling work is dominated by heuristic algorithms. The scheduling efficiency of the heuristic algorithm is not optimal. In this paper, we improved the deep reinforcement learning algorithm for the cluster scheduling, which named DeepCM. Test results on the simulation data shows that the DeepCM is capable of improving the performance for job scheduling on the cluster. The slowdown could be improved from 2.248 to 2.235 in a environment of 3 machines. The fusion of internal baseline and external baseline could reduce the variations of the performance on different jobsets. The experimental results demonstrate that the deep reinforcement learning get improved scheduling efficiency in cluster computing. The performance advantage is more obvious when the load gets heavier.

Keywords: Deep reinforcement learning · Schedule · Cluster · Policy gradient · Fusion baseline

1 Introduction

In computing clusters, we can access the CPU, memory, storage, software and other resources of different physical machines through the network. Computing tasks could be completed more efficiently by efficient utilization of the resources in the clusters. The establishment, lease, and even maintenance of computing clusters are expensive. Therefore, efficient utilization of computing clusters is essential. For large clusters, a small increment in utilization efficiency can save millions of investment [2].

Task scheduling is to establish the mapping relationship between tasks and computing resources. Tasks and computing resources can have a one-to-many or

Supported by National key R & D program high performance computing project (2017YFB0203501).

many-to-one relationship. The essence of scheduling is a combinatorial optimization problem. The mapping relationship between tasks and resources is extremely complex, and finding the optimal task scheduling strategy is complex and difficult. The existing scheduling method is dominated by heuristic algorithms. Common heuristic algorithms include Shortest Job First (SJF), fairness-based algorithms [3,5], and resource matching-based algorithms (such as Tetris [4]). Park et al. suggested to take the runtime uncertainty into consideration, and gave an end-to-end strategy for job scheduling with uncertainty [10]. Heuristic algorithms have the advantages of easy understanding, easy implementation, and strong generalization, and are widely used in various distributed systems. However, due to the lack of in-depth analysis of tasks and resources, heuristic algorithms are not optimal. In some scenarios, the scheduling efficiency is low. Task scheduling in a distributed environment still has a lot of room for improvement, especially for some specific scenarios.

Reinforcement learning has been widely used for sequential decision making in an unknown environment, where the agent learns a policy to optimize a cumulative reward by trial-and-error interactions with the environment [14]. In the last decade, the introduction of deep learning technology has promoted the rapid development of reinforcement learning, and has achieved success in games [9,13] and other scenarios. Reinforcement learning can learn the experience during the interaction with the environment and make better decisions. Task scheduling itself is also a sequential decision-making process. A natural idea is whether reinforcement learning can be used to optimize task scheduling in a distributed environment. Mao et al. implemented a task scheduling algorithm using reinforcement learning, the entire cluster was simplified into a resource pool, and tasks were allocated to the resource pool [8]. Since the whole cluster is one resource pool, the key factor is to determine the order of the tasks. In the follow-up work, Mao et al. described task dependencies through Directed Acyclic Graph (DAG), and modeled it using graph convolutional neural networks. Then reinforcement learning was adopted to schedule the tasks [7]. Compared with the heuristic scheduling algorithms, reinforcement learning algorithm has achieved significant efficiency improvements on both simulated and real data sets, which verifies the effectiveness of reinforcement learning in task scheduling. Neither of the above two algorithms restricts the task flow and they are general scheduling algorithms. Another research direction is the scheduling of dedicated task streams, such as machine learning clusters. By modeling the convergence curve of the machine learning algorithms, Peng et al. can estimate the end time of various deep learning tasks accurately. Heuristic greedy strategy is used to allocate resources, the efficiency improved more than 60% [11]. In another algorithm DL2, Peng et al. used reinforcement learning, combined with off-line initialization training and on-line learning for training, and the performance was improved by 17.5% [12]. Bao et al. tested the degree of interference between different machine learning tasks, used reinforcement learning for task scheduling, which tried to put tasks with low mutual interference in one computing unit (one or several machines). Experimental results show that compared with the traditional heuristic scheduling algorithm, the performance of this algorithm is improved by more than 25% [1]. The results showed that reinforcement learning algorithms improve the performance of machine learning clusters. Wang et al. applied a deep-Q-network model in a multi-agent reinforcement learning setting to guide the scheduling of multi-workflows over infrastructure-as-a-service clouds, which shows better performance than traditional scheduling method [16].

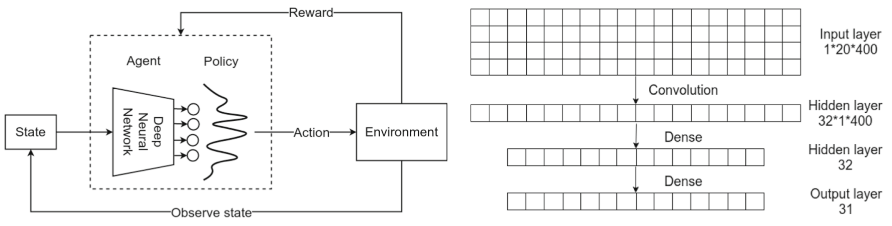
In Mao's works [7,8], the entire cluster is abstracted into a resource pool without considering the boundaries of physical machines. In this paper, we follow their method, the main contributions of this paper are summarized as follows:

- Applying reinforcement learning-based scheduling algorithms to more practical clusters, which taking resource boundaries of physical machines into consideration, aiming to improve resource utilization of the computing cluster.
- We propose a fusion baseline strategy, which adopt a fusion of internal baseline and external baseline as the final baseline in policy gradient algorithm, to reduce the variations of the reinforcement learning model.

The remainder of this paper is organized as follows. Section 2 presents our deep reinforcement learning model for job scheduling on cluster, Deep Cluster Management (DeepCM), in detail. Section 3 covers the experimental results on a simulation environment. Finally, Sect. 4 gives conclusions and discusses future work.

2 Deep Reinforcement Learning for Job Scheduling on Cluster

As other Reinforcement Learning (RL) system, the Deep Reinforcement Learning system for Cluster management include 3 important parts: environment(represented by state), agent and reward, which are illustrated in Fig. 1(a).



(a) Overall architecture of DeepCM (b) Architecture of the neural network

Fig. 1. System architecture of DeepCM and the architecture of the policy network.

2.1 RL Formulation for Schedule

In the application for job scheduling on cluster, the environment is the information about the jobs and the cluster. The state of the environment including the state of all the jobs that needed to schedule, and all the available resources on the nodes in the cluster. Suppose the cluster is composed of N nodes, each node contains D types of limited resources (CPU, memory, IO and disk etc.). As for the task, we suppose the resource and time requirement are known. The scheduling task is to allocate required resources of appropriate node to the jobs. **State**. The state of the system is represented by a big image composed of subimages. The subimages represent the state of candidate jobs, and nodes, and some extra information about job arriving time, and number of jobs out of the candidate queue. The image presentation of the system state is illustrated in Fig. 2. As show in the figure, the whole state image is composed by three kinds of subimages: subimage of machines, subimage of jobs and subimage of extra information. The state matrix representation is explained as follows.

1. Machine state image. Each machine image contains D images of different resources. In Fig. 2, one machine image of one resource was given for example. The size of subimage is $T \times K$, indicating the occupancy state of K units of resource in T timesteps. Each column of the image represents one unit of the resource, and each row represents a time step. The image contains the information about resource occupancy of the machine. The colored grid indicates that the resource has been occupied, which is represented by 1. The grids of the same color indicate that they are occupied by the same task. The white grids represent spare resources, represented by 0. The machine image is capable of representing the resource utilization of the machines. The size of one machine image is $T \times (D \times K)$
2. Job state image. A job stage image contains D subimages of different resources. The size of each subimage is $T \times K$, too. The colored grid indicates the resource and time the job required. For example, the job subimage in Fig. 2, the red square containing 2×2 grids, indicating executing this job require 2 units of this resource, and it will last for 2 timesteps. It should be pointed out that the amount of various resources required for the same job is different. For example, one job may need 3 units of CPU, and 1 units of memory. However, the duration of source requirement is the same. The size of one job image is $T \times (D \times K)$
3. Extra information state image. This state image of extra information contains information about the jobs out of the candidate job queue. The size of extra information image $T \times K$. The first $K - 1$ columns is about the number jobs in the backlog queue, which is the job that has been submitted but not in the candidate queue (Usually because the candidate queue is full). The number of rows with a value of 1 is proportional to the number of jobs in the backlog. The length of backlog is set to L_b, if N_b jobs in backlog, then the elements of the first $\lceil \frac{N_b}{L_b} \times T \rceil$ rows are set to 1. The blue part of the extra subimage indicates about 60% of the whole back queue are full. The last column is the number of timesteps past since last submitted job. The value of the whole column is $max(\frac{T_p}{T_w}, 1)$, where T_p is the number of timesteps past since last submitted job, T_w is a parameter.

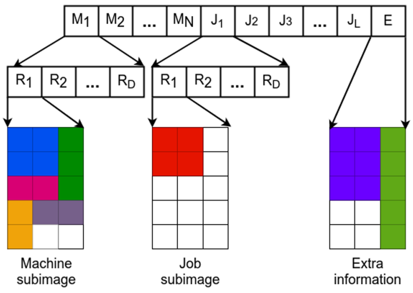


Fig. 2. Image representation of the state

Reward. Our goal is to minimize the average job slowdown. The job slowdown is defined by

$$S_j = \frac{C_j}{T_j}, \tag{1}$$

where C_j is the completion time of the job, which last from job submission to job completion, including four parts: the waiting time for entering the candidate queue, time in the candidate queue, waiting time on the scheduled machine, and time for execution. T_j is the ideal (minimum) duration of the job, including the time for execution only. It is easy to find that, $C_j > T_j$, so $S_j > 1$. The reward of each timestep is a piece of the negative slowdown, which defined as:

$$R_t = \sum_{j \in J_t} -\frac{1}{T_j}, \tag{2}$$

where J_t indicates all the jobs in the system at timestep t, including the job waiting for entering the candidate queue, the job in the candidate queue, the job scheduled on a machine (no matter it is waiting for execution or executing). The final cumulative reward is

$$R_c = \sum_{t=0}^{T_s} R_t = \sum_{t=0}^{T_s} \sum_{j \in J_t} -\frac{1}{T_j} = \sum_{j \in J} C_j \times (-\frac{1}{T_j}) = \sum_{j \in J} -\frac{C_j}{T_j}, \tag{3}$$

where R_c is the total cumulative reward of all the tasks in the T_s steps, J is the jobset of all the jobs during the T_s timesteps. The average job slowdown is defined as

$$R_a = -\frac{R_c}{|J|}, \tag{4}$$

where $|J|$ is the number of jobs in jobset J.

Agent. A reasonable schedule method should take multiple actions in one timestep. For simplicity, multiple action selection was achieved in a manner of repeat single action selections, only one action is selected at a time. Repeat the selection until a void action was selected. Suppose there are M candidate jobs and N machines. For each single action selection, we select a job (M candidates) and allocate a machine (N candidates), which leads to $M \times N$ selections and one void selection. There are $M \times N + 1$ selections in the action space for a single action selection, which is also the output dimension of the policy neural network shown in Fig. 1(a).

As shown in Fig. 1(a), the action policy is achieved by a neural network. The input is the state of the system, and the output is a probability distribution about potential actions. Figure 1(b) shows the architecture of the neural network, including the input layer, 2 hidden layers and 1 output layer.

Suppose there are 3 machines with 2 types of resources, each machine contains 10 units of either resources. At time step t, we can schedule job to t+20 timesteps. Under this setting, the number and size of the each layer are given in Fig. 1(b). The input layer has one input only, whose size is 20×400. The output of the convolution layer includes 32 feature maps with size of 1×400. The hidden layer as 32 neurons, with a dropout rate of 0.2. The output layer has 31 neurons with softmax activation, indicating the probability of 31 candidate actions.

2.2 Policy Gradient

We train the policy network with policy gradient descent algorithm [15]. As the name suggests, policy gradient means gradient descent of policy network. The policy neural network can be seen as a multi-class neural network. Its loss function is categorical cross-entropy. The goal of reinforcement learning is to maximize the expected cumulative reward, whose gradient is

$$\bigtriangledown E_{\pi_\theta}\left[\sum_{t=0}^{T}\gamma^t r_t\right] = E_{\pi_\theta}\left[\sum_{t=0}^{T}\bigtriangledown_\theta \log \pi_\theta(s,a) Q_\theta^\pi(s_t,a_t)\right], \tag{5}$$

where $Q_\theta^\pi(s_t,a_t)$ is the expected reward of choosing action a_t in state s_t. Its unbiased estimation obtained by Monte Carlo simulation [6] is denoted by $v_t = \widehat{Q_\theta^\pi(s_t,a_t)}$ and substitute v_t in (5), we have

$$\bigtriangledown E_{\pi_\theta}\left[\sum_{t=0}^{T}\gamma^t r_t\right] = E_{\pi_\theta}\left[\sum_{t=0}^{T}\bigtriangledown_\theta \log \pi_\theta(s_t,a_t) v_t\right], \tag{6}$$

The gradient can be decomposed into two terms, $\bigtriangledown_\theta \log \pi_\theta(s_t,a_t)$ is the gradient of conventional classification neural network, and v_t is the weight. As mention above, v_t is the expected reward of each step, we can find that the policy gradient is the conventional gradient weighted by reward. This is the key idea of policy gradient.

The policy gradient training algorithm was shown in Algorithm 1. There are J jobsets for training, each jobset run K episodes, which leads to K trajectories. Expected cumulative reward are estimated from the trajectories.

The cumulative reward has cumulative effect, which leads to heavy dependency on its time step. In order to reduce the significant difference caused by the position, a common trick of policy gradient algorithm is to adjust the rewards

Algorithm 1. Policy gradient training algorithm for cluster management

```
1: Initialize $\overrightarrow{b_e} = \overrightarrow{0.0}$
2: for each iteration do
3:     for each jobset $j = 1$ to $J$ do
4:         for episode $k = 1$ to $K$ do
5:             $\{s_1^k, a_1^k, r_1^k, s_2^k, a_2^k, r_2^k, ...s_{L_k}^k, a_{L_k}^k, r_{L_k}^k\} \sim \pi_\theta$
6:             Calculate returns: $v_t^k = \sum_{i=t}^{L_k} \gamma^{i-t} r_i^k$
7:         Calculate internal jobset baseline
               $$\overrightarrow{b_a^j} = \frac{\sum_{k=1}^{K} \overrightarrow{v^j}}{K} \quad (7)$$
8:         Calculate jobset baseline
               $$\overrightarrow{b^j} = \beta \overrightarrow{b_a^j} + (1-\beta) \overrightarrow{b_e} \quad (8)$$
9:     Calculate the gradient of all the samples.
               $$\triangle\theta = \bigtriangledown_\theta \log \pi_\theta(s,a)(v-b) \quad (9)$$
10:    Calculate external jobset baseline
               $$\overrightarrow{b_e} = \frac{\sum_{i=1}^{J} \overrightarrow{b_a^j}}{J} \quad (10)$$
11:    $\theta \leftarrow \theta + \alpha \triangle \theta$
```

by subtracting the baseline, which is also used in [8]. The conventional baseline calculation is shown in (7), which is the average of different trajectories of the same jobset. We name it internal jobset baseline. The trajectories subtract the baseline as the reward. In the training process, we found that the differences between jobsets are also very large, and internal baseline adjustments cannot handle the difference.

We use the weighted average of the internal baselines of different jobsets as the external baseline between jobsets (10), and combine the external baseline in the last iteration and the internal baseline through linear combination (8). We use the combinational result as the final baseline for reward adjustment, expecting better learning performance. The introduction of an external baseline is a major improvement for policy gradient, and subsequent experimental results verify the effectiveness of this mechanism.

Gradients were calculate by (9), which is a cumulative gradient of all the samples in this iteration. The samples include all the trajectories of all the jobsets.

3 Experimental Results

This section contains the experimental results. First explain our test environment. Then compare the training procedure of the conventional and improved policy gradient model. Compare the schedule efficiency of our model with referenced models under different settings.

3.1 Environment Setting and Reference Models

In our setting, there are 1–3 machines in the cluster, each machine has 2 types of resources, both are divided into 10 units. Tasks are randomly generated as a batch, according to the same distribution, 20% tasks are long tasks, whose duration is uniformly from 10 to 15 timesteps. 80% of the tasks are short tasks, whose duration is from 1 to 3 timesteps. All the tasks have one dominant resource that is randomly selected. The dominant resource requirement is uniformly from 4 to 6 units of resource, and the other resource requirement is uniformly from 1 to 2 units of resource. The tasks submitted to the cluster with a probability of p_s, which is used for workload control. In the following experiments, we suppose one jobset contains 50 tasks for both training and testing.

The parameters involved in the training algorithm include: The discount parameter during cumulative reward calculation is $\gamma = 1.0$. The fusion parameter of internal baseline and external baseline β is set to 0.9. We train the network for 1000 or 1500 iterations, if the reward did not improve from the 900th to the 1000th iteration, it will stop at the 1000th iteration; Otherwise, it will stop at the 1500th iterations. Another important parameter is the learning rate, which is set to $lr = 0.003$ at the beginning, and recalculated every 30 iterations by $lr = \max(lr \times 0.8, 0.001)$.

We adopt 4 scheduling models as reference. Shortest Job First (SJF) is one of the best heuristic scheduling method. Take machines into consideration, there are two strategies for allocation: compact strategy, which allocate the job to the machine with highest resource utilization; and spread strategy, which allocate the job to the machine with lowest resource utilization. Combine SJF with compact and spread leads to SJF compact and SJF spread, together with Tetris [4] and DeepRM [8], there are the 4 reference models. Here, DeepRM was modified for cluster scheduling scenario.

Our DeepCM model has similar principle as DeepRM. However, the policy gradient algorithm is improved by a mechanism of fusion baseline, as mentioned in Algorithm 1.

3.2 Tests on Cluster with Different Number of Machines

In this part, we test the same workload on cluster with different number of machines. We set the workload control parameter $p_s = 0.8$. The number of machines is set to 1,2 and 3. Since 3 machines leads to almost no slowdown, we reduce each type of the resource on a machine from 10 units to 8 units.

The average slowdown is shown in Table 1. From the table, we can find that no matter how many machines are in the cluster, DeepCM and DeepRM achieve consistently better result than conventional heuristic methods. The performance of DeepRM is close to DeepCM. DeepCM is slightly better than DeepRM.

When there is only one machine, we have no choice, the 3 heuristic schedule models degrade to the SJF schedule. The average slowdown in the top 3 rows of the first column are the same.

Table 1. Average slowdown on cluster with different number of machines

Scheduling model	Average slowdown		
	1	2	3
SJF compact	2.833(±0.850)	1.260(±0.213)	1.147(±0.125)
SJF spread	2.833(±0.850)	1.263(±0.215)	1.149(±0.127)
Tetris	2.833(±0.850)	1.267(±0.201)	1.152(±0.130)
DeepRM	2.805(±0.846)	1.254(±0.206)	1.144(±0.128)
DeepCM	**2.773(**±0.839**)**	**1.245(**±0.178**)**	**1.143(**±0.125**)**

Compare the average slowdown on the same model with different number of machines, we find that the slowdown decreases as the number of machines increases. That is due to more resources are available as the number of machines increases. However, the performance improvement of DeepCM decrease as the machine number increase. When there is one machine, DeepCM outperforms the best heuristic model by 0.06, outperforms DeepRM by 0.032. When there are three machines, DeepCM outperforms the best heuristic model by 0.004, outperforms DeepRM by 0.001. We infer that as more machines get involved, resources are no longer tight, and the advantages of scheduling shrinks.

3.3 Tests Under Different Workloads

In this part, we fix the number of machines in cluster as 3, and test different workload on it. We set the workload control parameter p_s as $0.7, 0.8, 0.9$ and 1.0. Each machine contains 8 units of each type of resources.

The average slowdown is shown in Table 2. From the table, we can find that no matter what value p_s is, DeepCM and DeepRM achieve consistently better result than conventional heuristic methods. The performance of DeepRM is close to DeepCM. DeepCM is slightly better than DeepRM.

Compare the average slowdown on the same model with different workload, we find that the slowdown increases as the workload increases. The performance improvement of DeepCM did increase as the workload increase. When the control parameter is 0.7, DeepCM outperforms the best heuristic model by 0.002, outperforms DeepRM by 0.001. When the control parameter is 1.0, DeepCM outperforms the best heuristic model by 0.019, outperforms DeepRM by 0.006.

Combined with the previous experimental results in Table 1, we can conclude that DeepCM can achieve better performance than conventional models, especially when the workload is heavy. Moreover, the heavier the workload, the greater performance gain can be achieved.

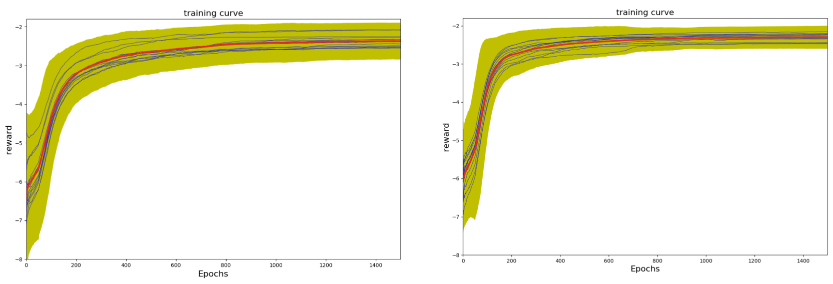
Table 2. Average slowdown of different workload

Scheduling model	Average slowdown			
	0.7	0.8	0.9	1.0
SJF compact	1.041(±0.042)	1.147(±0.125)	1.455(±0.265)	2.248(±0.133)
SJF spread	1.044(±0.050)	1.149(±0.127)	1.460(±0.264)	2.254(±0.139)
Tetris	1.045(±0.051)	1.152(±0.130)	1.459(±0.264)	2.254(±0.139)
DeepRM	1.040(±0.039)	1.144(±0.128)	1.450(±0.261)	2.241(±0.131)
DeepCM	**1.039(**±0.040)	**1.143(**±0.125)	**1.448(**±0.261)	**2.235(**±0.126)

Compared to scheduling on single resource pool, the scheduling on the cluster is more difficult, and we see that the improvement of DeepRM relative to the heuristic methods is small. DeepCM has further improved DeepRM. Although the numerical improvement is small, it is still important for cluster scheduling.

3.4 Benefits of the Fusion Baseline

We have seen that DeepCM has achieved better performance than conventional scheduling methods. In this section, we focus on the comparison between DeepCM and DeepRM, to demonstrate why the fusion baseline policy gradient works better than conventional policy gradient. We compared the training curve of the fusion baseline policy gradient and conventional policy gradient. Training curves of both algorithms are shown in Fig. 3. The red curve is the average slowdown of all the jobsets, the yellow region is 3σ (σ is the standard deviation) around the average curve, the blue curves are average slowdown of each jobset. The average slowdown is smoothed by moving average with 50 as the moving window width.



(a) Training curves of conventional PG (b) Training curves of FPG

Fig. 3. Training curves. The red curve is the average slowdown of all the jobsets, the yellow region is 3σ (σ is the standard deviation) around the average curve, the blue curves are average slowdown of different jobset. The average slowdown is smoothed by moving average. (Color figure online)

Table 3. Statistics of the training curves

		Max	Min	Mean
PG	Mean	–2.369	–6.444	–2.812
	Std	0.692	0.157	0.218
FPG	mean	–2.300	–6.067	–2.611
	Std	0.646	0.093	0.156

Figure 3(a) is the training curves of conventional policy gradient (PG), Fig. 3(b) is the training curves of the fusion baseline policy gradient (FPG). Statistics of the training curves are listed in Table 3. By comparison, we can find that:

1. Compare the red curve in both figures. It is easy to find that the mean of rewards increase sharply at the beginning. As the training iterations increase, the growth rate decreases. Quantitatively, the average reward of PG increases from -6.444 to -2.369, while that of FPG increases from -6.067 to -2.300.
2. The curve of FPG increases more sharply than PG, PG reach -3 at 265th iteration, while FPG reach the same value at 147th iteration. Which means FPG can be trained more efficiently.
3. The standard deviation of PG is larger than that of FPG. The standard deviation of PG are reduce from 0.692 to 0.157, with mean value of 0.218. The standard deviation of FPG are reduced from 0.646 to 0.093, with mean value of 0.156.
4. By observing the blue curves, we find that the average reward on all jobsets of PG has no significant improvement after 700 iterations; while that of FPG has three significant improvements after 700 iterations, which leads to smaller deviation.

4 Conclusion and Future Works

In this paper, we improved the reinforcement learning based scheduling algorithm, and applied it for the cluster resource management, which named DeepCM. Test results on the simulation dataset shows that DeepCM is capable of improving the performance for job scheduling on the cluster, especially when the workload is heavy. Furthermore, the fusion of internal baseline and external baseline could reduce the variation of the performance on different jobsets, which makes the trained model more steady.

However, it is worth noting that, as the number of candidate jobs or the number of machines increases, both the state representation and the output dimension would increase sharply, which leads to bad scalability of the DeepCM. Poor scalability limits the application of deep reinforcement learning scheduling algorithms to large-scale clusters. In our further work, we would try to enhance the scalability of DeepCM and related algorithm.

Future work will include improving the performance of the reinforcement learning by incorporating more layers into the policy network, and considering more complex application scenarios, such as the inter dependency of the jobs, the actual resource occupation of the jobs, and the uncertainty of runtime estimation. Another possible direction is multi-objective optimization, which optimize cost, make-span and delay, etc. simultaneously.

References

1. Bao, Y., Peng, P., Wu, C.: Deep learning-based job placement in distributed machine learning clusters, pp. 505–513 (2019)
2. Barroso, L.A., Hlzle, U.: The datacenter as a computer: an introduction to the design of warehouse-scale machines. Synth. Lect. Comput. Archit. **8**(3), (2009). https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/FairScheduler.html
3. Ghodsi, A., et al.: Dominant resource fairness: Fair allocation of multiple resource types. In: Proceedings of the 8th USENIX Symposium on Networked Systems Design and Implementation (NSDI), vol. 11, pp. 323–336 (2011)
4. Grandl, R., Ananthanarayanan, G., Kandula, S., Rao, S., Akella, A.: Multi-resource packing for cluster schedulers. ACM SIGCOMM Comput. Commun. Rev. **44**(4), 455–466 (2014)
5. Hadoop, A.: Hadoop fair scheduler. http://hadoop.apache.org/common/docs/stable1/fair_scheduler.html (2014)
6. Hastings, W.K.: Monte carlo sampling methods using markov chains and their applications. Biometrika **57**(1), 97–109 (1970)
7. Mao, H., Schwarzkopf, M., Venkatakrishnan, S.B., Meng, Z., Alizadeh, M.: Learning scheduling algorithms for data processing clusters. In: Proceedings of the ACM Special Interest Group on Data Communication, pp. 270–288 (2019)
8. Mao, H., Alizadeh, M., Menache, I., Kandula, S.: Resource management with deep reinforcement learning. In: Proceedings of the 15th ACM Workshop on Hot Topics in Networks, pp. 50–56 (2016)
9. Mnih, V., et al.: Human-level control through deep reinforcement learning. Nature **518**(7540), 529–533 (2015)
10. Park, J.W., Tumanov, A., Jiang, A., Kozuch, M.A., Ganger, G.R.: 3sigma: distribution-based cluster scheduling for runtime uncertainty. In: Proceedings of the Thirteenth EuroSys Conference (2018)
11. Peng, Y., Bao, Y., Chen, Y., Wu, C., Guo, C.: Optimus: an efficient dynamic resource scheduler for deep learning clusters. In: Proceedings of the Thirteenth EuroSys Conference, pp. 1–14 (2018)
12. Peng, Y., et al.: Dl2: A deep learning-driven scheduler for deep learning clusters. In: arXiv preprint arXiv:1909.06040 (2019)
13. Silver, D., et al.: Mastering the game of go with deep neural networks and tree search. Nature **529**(7587), 484–489 (2016)
14. Sutton, R., Barto, A.: Reinforcement Learning: An Introduction. MIT Press, Cambridge (1998)
15. Sutton, R.S., Mcallester, D., Singh, S., Mansour, Y.: Policy gradient methods for reinforcement learning with function approximation. In: Advances in Neural Information Processing Systems, pp. 1057–1063 (1999)
16. Wang, Y., et al.: Multi-objective workflow scheduling with deep-q-network-based multi-agent reinforcement learning. IEEE Access **7**, 39974–39982 (2019)