Adaptive Online Estimation of Thrashing-Avoiding Memory Reservations for Long-Lived Containers

Jiayun Lin[1], Fang Liu[1(✉)], Zhenhua Cai[1], Zhijie Huang[1], Weijun Li[2], and Nong Xiao[1]

[1] School of Data and Computer Science, Sun Yat-Sen University, Guangzhou, Guangdong 510275, China
{linjy57,caizhh8}@mail2.sysu.edu.cn, {liufang25, xiao6}@mail.sysu.edu.cn, jayzy.huang@gmail.com
[2] Shenzhen Dapu Microelectronic Co., Ltd., Shenzhen, China
liweijun@dputech.com

Abstract. Data-intensive computing systems in cloud datacenters create long-lived containers and allocate memory resource for them to execute long-running applications. It is a challenge to exactly estimate how much memory should be reserved for containers to enable smooth application execution and high resource utilization as well. Current state-of-the-art work has two limitations. First, prediction accuracy is restricted by the monotonicity of the iterative search. Second, application performance fluctuates due to the termination conditions. In this paper, we propose two improved strategies based on MEER, called MEER+ and Deep-MEER, which are designed to assist in memory allocation upon resource manager like YARN. MEER+ has one more step of approximation than MEER, to make the iterative search bi-directional and better approach the optimal value. Based on reinforcement learning and rich data, Deep-MEER achieves thrashing-avoiding estimation without involving termination conditions. Based on the different input requirements and advantages, a scheme to adaptively adopt MEER+ and Deep-MEER in cluster life cycle is proposed. We have evaluated MEER+ and Deep-MEER. Our experimental results show that MEER+ and Deep-MEER yield up to 88% and 20% higher accuracy. Moreover, Deep-MEER guarantees stable performance for applications during recurring executions.

Keywords: Memory reservation estimation · Intelligent cluster scheduling · Cloud datacenter · Reinforcement learning · Long-live container

1 Introduction

With the recent technical breakthroughs in big data, a dramatically increasing number of heterogeneous in-memory computation workloads, including machine learning [13–15], streaming processing [16–18], interactive query [19–21], and graph computation [22–24], are being deployed on shared cluster in data centers. These kinds of workloads

H. Gao et al. (Eds.): CollaborateCom 2020, LNICST 349, pp. 620–639, 2021.
https://doi.org/10.1007/978-3-030-67537-0_37

dealing with huge amounts of data are called long running applications (LRAs) [11], which have become the dominant workloads.

In today's data centers, clusters rely on resource managers, such as YARN [5], Mesos [6], Omega [7], Borg [8] and Kubernetes [9], to allocate resources for applications. These managers schedule computing resources like CPU and memory by packaging them as "containers". Different from traditional short-lived containers built to deal with batch jobs, containers for LRAs stay alive until an application completes, so it is named "long-lived containers".

Since the computing resources are limited and valuable, a long-existing concern that still remains in the industry is how to quantify user's resource demand when building a long-lived container. Take memory for instance, if too much memory is allocated to an application, the application utilizes only part of the allocated memory and the rest is wasted. On the contrary, if too little memory is reserved, the performance of the application may not be guaranteed and more seriously, the application may crash due to lack of memory. The optimal memory reservation estimation issue becomes more challenging based on this context.

1.1 Performance Inflection Point and Motivation

From previous works, we learn the concept of memory reservation elasticity [10] that slight reduction of memory supply under over-provision has no serious impact on performance. This property benefits from memory manager's two protection mechanisms. When the memory is insufficient, garbage collection will be triggered to free memory, or the program will be spilt to the disk. Under this situation, the slight performance penalty caused by insufficient memory supply is acceptable.

To understand memory elasticity, we submit the same application under different memory reservation and record the execution time. We use four different workloads to prove that memory reservation elasticity is a universal phenomenon. The result is shown as Fig. 1. It can be clearly observed that the curve has a long tail. Under over-reservation, moderate reduction on memory reservation has almost no effect on performance. But when memory reservation drops to a certain point, even slight reduction of memory leads to a sharp performance degradation. We call this point "performance inflection point", which is circled in red. Let's describe performance inflection point in detail using formula. We sort reserved memory sizes from large to small, denote them by $M_0, M_1, \ldots, M_n$ and the corresponding execution time by $T_0, T_1, \ldots, T_n$, and then compute the average of the leading execution time

$$T^i_{avg} = \frac{1}{i} \sum_{n=0}^{i-1} T_n \tag{1}$$

for each i, i = 1, . . . , n. There exists a set of index I that for each i ∈ I, T_i satisfies the following formula:

$$\frac{T_i - T^i_{avg}}{T^i_{avg}} > 0.2. \tag{2}$$

The minimum value in the set minus one is the subscript corresponding to the performance inflection point, which is denoted by M in this paper. We have tried a lot of

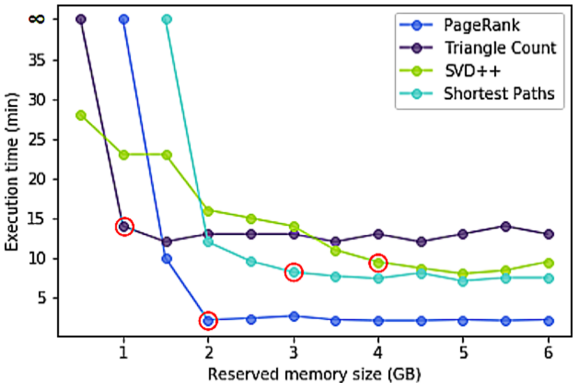

Fig. 1. Application execution time under various memory reservation.

minima. When it is set as 0.2, the calculated inflection points are most in line with people's intuitive feelings. Once we find the performance inflection point, we can find the best balance between memory utilization and application performance and this inflection point can be considered as optimal memory reservation.

A leading technique to estimate optimal memory reservation is MEER [10]. MEER uses histogram analysis to learn the rule of memory occupation and estimates an optimal memory reservation through iterative search. Although this work shows good results on estimation accuracy and saving memory resource, there is still room for improvement:

1. **The iterative search of MEER is monotonous.** MEER has the characteristic of self-decay, i.e., the estimated result decreases progressively and irrevocably before termination, which limits the opportunity for correction. Accordingly, MEER+ turns iterative search into bi-directional by adding a step of approximation. It achieves excellent accuracy in case that there is no enough historical data for detailed analysis and forecasting.
2. **Application performance is thrashing.** Both MEER and MEER+ have to experience inefficient execution required by the termination conditions, which may not be accepted by users. Deep-MEER exploits the potential of historical data generated in application execution with the help of reinforcement learning. Leveraging this unsupervised learning technique for data analysis, Deep-MEER wriggles out of termination conditions.

In the process of overcoming these limitations, we discovered an adaptive scheme to adopt MEER+ and Deep-MEER flexibly in different cluster life time to take full advantage of their own strengths: applying MEER+ to seek out optimal memory reservation quickly, then using thrashing-avoiding Deep-MEER for sustainable development of clusters as soon as reinforcement learning-based model is well trained with abundant historical data.

1.2 Main Contributions

The main contributions of this paper are as follows:

- To achieve higher estimation accuracy, we propose MEER+, which refines step size of iterative search to rectify the estimation result. MEER+ has an additional process of bidirectional finer search, where estimated reservation is tuned up by a more precise step size to approach the optimal reservation.
- To protect applications from performance thrashing, we propose Deep-MEER, which fully utilizes historical data by applying reinforcement learning-based estimation algorithm. Deep-MEER is not subject to termination conditions, so it avoids sever performance loss caused by the termination conditions.
- We show how to adopt MEER+ and Deep-MEER adaptively in cluster life cycle. Specifically, MEER+ makes a preliminary estimation when the cluster lacks data for reference in its infancy, while Deep-MEER enables stable application performance when the cluster has enough historical data in its maturity.
- We evaluate MEER+ and Deep-MEER on a cluster. Compared with MEER, MEER+ and Deep-MEER increase estimation accuracy by 88% and 20% respectively, and outperforms in resource utilization by 2×. Deep-MEER also has good generalization ability and incurs a smoother performance fluctuation curve.

The rest of this paper is organized as follows. Section 2 discusses the related work. Section 3 introduces accurate MEER+ and Sect. 4 describes thrashing-avoiding Deep-MEER, respectively. Further on, we present an adaptive strategy in Sect. 5. Section 6 gives the experimental results and the final section concludes this paper.

2 Related Work

Resource demand and system configuration forecasting has always been a research hotspot of resource management in datacenters. One of the most appealing research directions and challenges is that how resource elasticity can be leveraged to achieve trade-off between resource and efficiency. [25] first put forward the concept of memory elasticity in task level and proposes YARN-ME that leverages memory elasticity to reduce memory, which in turn reduces waiting time and makes task execution faster. Elasecutor [7] is present to be an elastic scheduler that dynamically allocates and explicitly sizes resources to executors. Then [10, 29] explore the same property in application level, which aims at finding out inflection point of application performance to make optimal memory reservation. More recently, Pufferfish [30] is a new manager to realize memory elasticity via lightweight virtualization.

People used to adopt default parameters or adjust parameters according to empiricism, which is rigid, manual and usually not the best. What algorithm has the best forecasting ability and what factor is the most influential is a problem worthy of discussion. MEER [10] and Prometheus [29] apply histogram analysis algorithm. Selecta [31] uses latent factor collaborative filtering to determine near-optimal configurations for cloud compute and storage resources. CherryPick [27] leverages Bayesian Optimization to build performance models for various applications. Elasecutor [32] relies

on support vector regression (SVR) to train the model that predicts demand time series. When configuring large-scale systems with complex parameters, it is not wise to rely just on empiricism. With the rise of machine learning, scholars begin to consider how this adaptive and automated method can assist in decision making in the field of system parameter tuning [1, 2, 12]. Radical basis function neural network (RFBNN) and Swarm Intelligence Based Prediction Approach (SIBPA) [35] are developed to address the various complex factors in resource allocation. In [34], novel workload latent features are proposed and computed by applying unsupervised learning on the access logs. Deep dueling [28] is an advanced deep learning architecture that attains the optimal average reward much faster than Q-learning and is more applicable to resource slicing problem. Resource demand and system configuration forecasting algorithms integrated with machine-learning technique may be a major research status in the future.

There are various types of data and workloads [13–24] in data centers. In face of such diversity, how to optimize resource allocation adaptively for specific scenarios becomes an important concern. BestConfig [4] achieves automate configuration tuning under given application workloads, which shows improvement of performance on a diversity of workloads. Elastisizer [26] addresses cluster sizing problem automatically for different data-processing jobs. This issue remains to be further explored.

3 Estimation Under Insufficient Historical Data

In this section, we will first describe the overall architecture of MEER, point out the monotonicity of MEER, and then describe our design of MEER+ for achieving better estimation precision with one more step of approximation.

3.1 MEER Overview

MEER is a system that assists schedulers to efficiently estimate optimal memory reservations for diverse in-memory computing workloads. As seen in Fig. 2, MEER includes two stages:

Stage 1. Pilot Run (Blue Line). When the application is submitted for the first time, it runs under over-reservation M_0. This stage generates the original memory footprints for the following stage, which are recorded by the history server and the metrics system. Then MEER figures out the expectation of memory usage M_1 from memory footprints by using histogram analysis model and informs resource manager of the result.

Stage 2. Iterative Search (Red Line). When the application is submitted for the n th time, resource manager adopts the last estimated result M_{n-1} as memory reservation of this execution. Similarly, MEER records memory footprints and computes the expectation of memory usage M_n. Next, it evaluates the performance for judgement of termination. If the performance meets one of the termination conditions, iterative search is terminated and M_{n-2} is the ultimate estimated optimal reservation, which is the minimum estimated result that ensures application performance. Otherwise, MEER informs resource manager of the computed expectation M_n for next submission. In general, there are three termination conditions: 1) execution time is too long, 2) garbage collection is

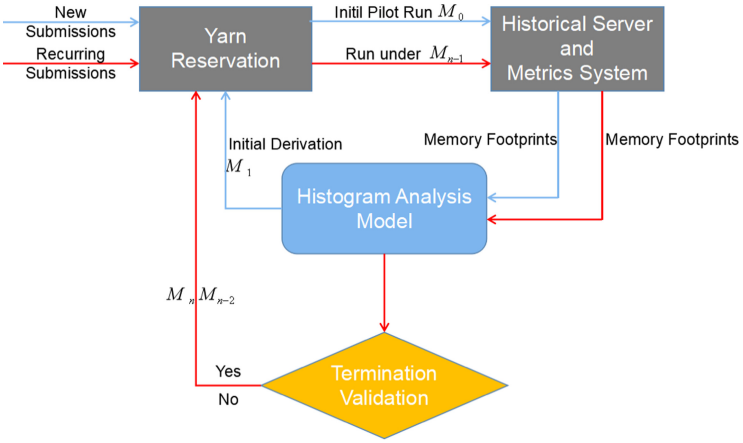

Fig. 2. Architecture and workflow of MEER. (Color figure online)

too time-consuming, and 3) memory utilization is satisfying. Except the third condition, for an iterative search to terminate, application must go through a time-consuming and inefficient run.

Limitation of Monotonicity. It is not hard to find that there is a certain gap between the optimal memory reservation estimated by MEER and the ground truth. We carefully observe the change of reserved memory size in the operation of MEER and find that in the whole process, the memory only decreases but not increases as shown in Fig. 3. Once the estimation result exceeds the inflection point, it stops immediately. This characteristic of monotonicity limits its opportunity of further approximation.

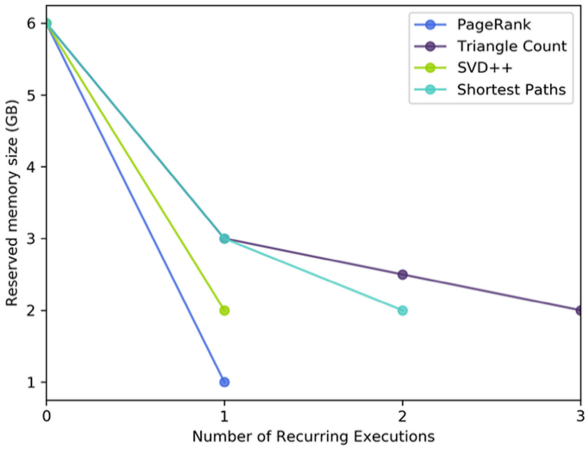

Fig. 3. Monotonic variations of memory reservation during recurring executions.

3.2 Design of MEER+

Performance inflection point divides memory into three categories: (1) over-reservation, (2) under-reservation, and (3) optimal reservation. The first includes values larger than optimal reservation M while the second includes values less than M. Experimental results show that M_{n-1}, which meets termination condition and stops iterative search, is either the first or the second type as shown in Fig. 4. In most cases, M_{n-1} is less than M. This is because MEER usually stops when it meets sharp fluctuations in performance out of insufficient memory reservation. In this case the termination condition met is time-consuming execution time or garbage collection. The only exception is Triangle Count. In this workload, iterative search is stopped because the memory usage expectation is close to reserved memory. Memory is already made full use of so there is no need to continue the search anymore. In this case, the termination condition met is good memory utilization.

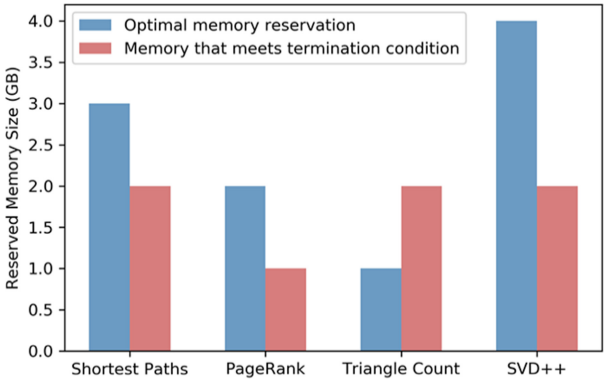

Fig. 4. Optimal memory reservation vs. Memory reservation that meets termination condition in MEER

MEER+ narrows the gap between M_n and M by destroying its monotonicity. It has an additional stage of approximation. It has been proved that M_{n-1} is either less than M or larger. Correspondingly, MEER+ reduces the gap by increasing or decreasing M_{n-1} within appropriate range. In approximation stage, the optimal reservation estimated by MEER+ is defined as:

$$\begin{cases} M_n = M_{n-1} + M_f, M_t < M \\ M_n = M_{n-1} - M_f, M_t > M \end{cases} \tag{3}$$

where M_t is the estimated memory reservation that meets termination condition and M_f is the increment or decrement added for rectifying the estimation result. Professionally, we call M_f step size. In case that $M_t < M$, since $M_t < M < M_{t-1}$, M_n is bound to nearer to M as long as:

$$M_f < M_{t-1} - M_t \tag{4}$$

And in case that $M_t > M$, M_n is certainly nearer to M as long as:

$$M_f < M_t - M \tag{5}$$

In a word, the estimation precision must be improved as long as the step size is appropriately set.

The workflow of MEER+ is shown in Fig. 5. There are three stages in total.

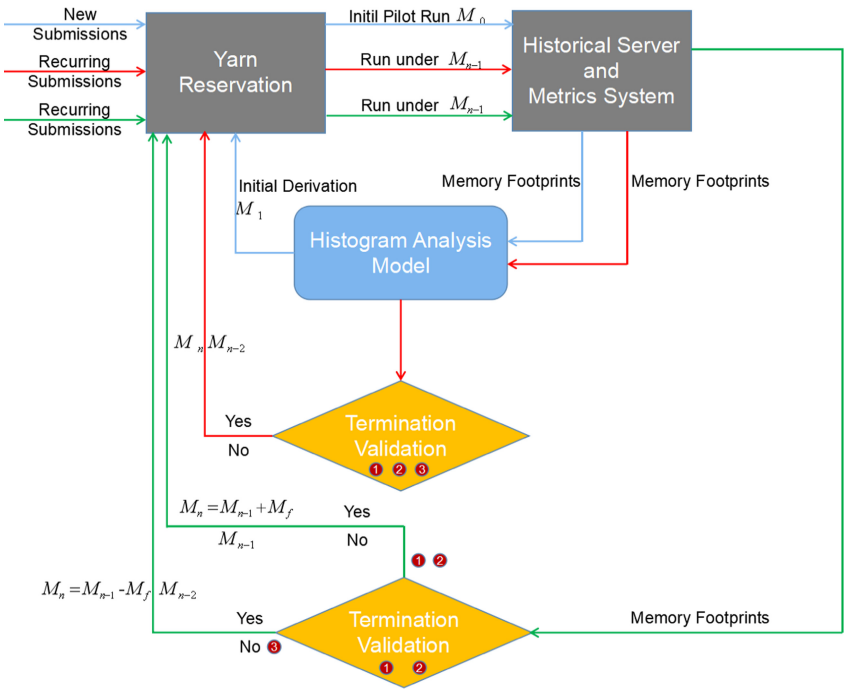

Fig. 5. Architecture and workflow of MEER+ (Color figure online)

Stage 1. Pilot Run (blue line). The same as Stage 1 of MEER.
Stage 2. Iterative Search (red line). The same as Stage 2 of MEER.
Stage 3. Approximation (green line). There is two branches according to termination conditions met in Stage 2. If condition 1 or 2 is met, Stage 3 is terminated if neither

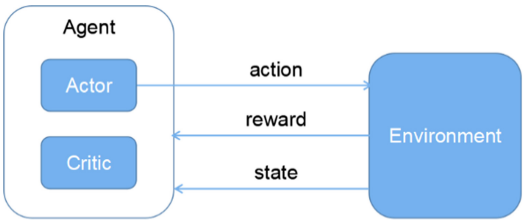

Fig. 6. Framework of actor-critic learning.

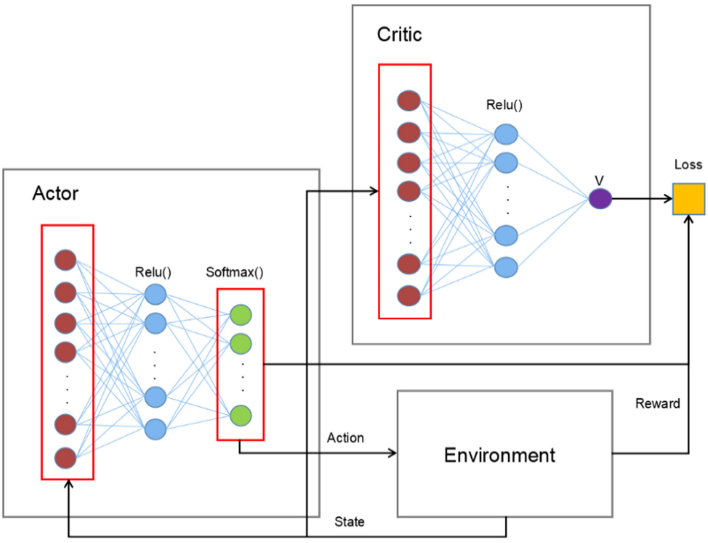

Fig. 7. Architecture of Actor-Critic learning in Deep-MEER.

termination condition 1 nor 2 is satisfied, and M_{n-1} is the ultimate estimated optimal memory reservation, which is the minimal memory reservation making the application runs normally again. Conversely, if condition 3 is met, MEER+ doesn't stop until condition 1 or 2 is met. Under this circumstance, M_{n-1} first leads to performance degradation so MEER+ selects M_{n-2} as the ultimate estimated optimal memory reservation.

4 Estimation Under Sufficient Historical Data

In this section, we will focus on performance stability. We will first describe the overall architecture of Deep-MEER, and then the design of reinforcement learning model and parameters for achieving thrashing-avoiding optimal memory reservation estimation with historical data.

4.1 Design of Deep-MEER

Limitation of Performance Trashing. When using MEER+ to look for optimal memory reservation, the application always suffers from a sharp performance degradation because termination is indicated by time-consuming execution or garbage collection. This may not be accepted by users in big data service.

Deep-MEER makes full use of the abundant historical data generated in MEER+ to eliminate the effect of termination conditions on application performance. Figure 8 shows the workflow of Deep-MEER. The estimation model is replaced by a reinforcement learning-based actor-critic learning model. Every time a submission comes, resource manager allocates memory resource with the model's guidance. Memory footprints

generated are then used to further train the model. As we can see, the estimation is never limited by termination conditions. It is able to stop before the estimation result exceeds the performance inflection point so that users will never experience bad application performance.

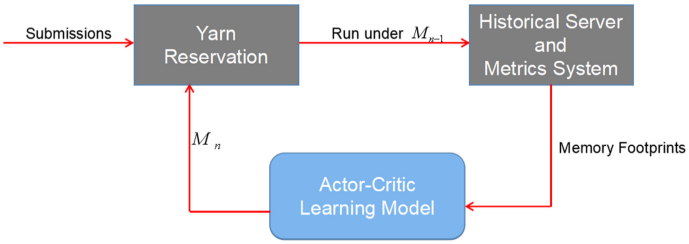

Fig. 8. Architecture and workflow of Deep-MEER.

4.2 Model Structure

Deep-MEER applies reinforcement learning by regarding the cluster as environment and seeing the memory estimator as agent. As shown in Fig. 6, the interaction between agent and environment is realized by the mutual transfer of three parameters: state, reward and action. Agent makes decisions under the guidance of a policy, i.e., provides action for the environment. Environment performs the action and responds to agent by returning reward and state. Then the agent learns from the return values, improves the policy to maximize the accumulated rewards. In actor-critic learning, agent is further separated into two units:

Actor. The actor acts as policy. It chooses an action based on probability and modifies the probability of each action according to the reward from environment and the judgement provided by critic. It implements a stochastic policy that maps the system state to the corresponding action.

Critic. The critic acts as value function that evaluates the policy. It passes judgement on the chosen action about how much it benefits the environment and provides useful reinforcement feedback for adjusting the policy.

More narrowly, actor and critic are both implemented by neural network. The whole structure is shown in Fig. 7.

Actor. Actor consists of three layers: input layer, hidden layer and output layer. The inputs of input layer are states coming from environment. Each circle represents a neuron and each line between two neurons represents a weight. Let ω denote the set of such parameters of weight. The value of each neuron is the weighted sum of the neurons in the previous layer except that the value of input neurons is provided by the environment. The mapping of neurons between each two layers is linear, which limits the expression ability of the model. Therefore, we introduce nonlinear factors by activation function to make

the neural network approach any nonlinear function at well. The activation function of hidden layer is Relu(). The neurons in output layer correspond to the action that how much memory reservation is set for this application. They finally go through a Softmax() function, with the help of which the outputs are transformed to values between 0 and 1 and the sum of them is equal to 1. In other words, each output stands for the probability that the action should be chosen. The greater the probability, the more likely it is that the action will bring the greatest benefit.

Critic. The basic structure and parameter form of critic are similar to that of actor. The only difference is that the output layer has only one neuron which is the score given by the critic. Once it figures out the value, the value is combined with reward from environment to finally compute the loss, which will be used to guide the actor and critic to update.

Environment. Environment is the cluster to run the application. On the one hand, it runs the submitted application under memory reservation determined by the actor. On the other hand, it returns the state changed after executing the action as well as the reward that measures how much it benefits form the action.

Algorithm 1 Actor-Critic Model-based Estimation Algorithm

1: Initial state S_0
2: Operate the critic unit to compute $V(S_0)$
3: **for** $i = 1, 2, ..., N$ **do**
4: Operate the actor unit to compute probability P_{i-1} of each action based on S_{i-1} and determine A_{i-1} with the highest probability
5: Execute A_{i-1} to obtain S_i and R_i
6: Operate the critic unit to compute $V(S_i)$
7: Compute TD error δ_{i-1}
8: Compute the loss $Loss(\delta_{i-1})$
9: Update parameters ω of actor and critic guided by the loss
10: **end for**

Algorithm 1 is the pseudo code of the actor-critic model-based estimation algorithm. S_t, A_t and R_t refer to state, action and reward at time t respectively. MEER+ starts with running application under over-reservation. Accordingly, when initialing S_0, DeepMEER reserves excessive memory. TD-error in the 7th line is commonly used to adjust the policy, which is defined as:

$$\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t). \tag{6}$$

Loss function in the 8th line can be selected freely according to individual needs. Updating parameters in the 9th line refers to that following the chain rule, neural network operates back propagation that calculates the derivative of composite functions, propagates the gradient of the output unit back to the input unit, and adjusts the learnable parameters of the network according to the calculated gradient.

4.3 Model Parameters

We need to define the interaction information parameters to make the model run.

State. Table 1 shows states that are provided by the environment.

Table 1. Environment state parameters.

States	Meaning
Δt	Increased execution time compared to initial run
E	Expectation of memory footprints
$max_1 \sim max_n$	The n most frequent value of memory footprints in histogram analysis
$p_1 \sim p_n$	Corresponding frequency of $max_1 \sim max_n$

Reward. Reward is used to determine how good it is to execute a given action under the given state. The target of our model is to save memory and remain good performance at the same time. Therefore, the more memory is saved and the less time it takes to complete the application, we are closer to our expectations. Here we define reward at time t as a function:

$$R_t = \frac{M_0 - M_t}{T_t - T_0 + 1}, \tag{7}$$

where M_t and T_t refer to reserved memory size and execution time at time t respectively. We add one to the denominator to prevent the molecule from being divided by zero.

Action. In terms of action, each neuron of actor's output layer corresponds to a specific memory setting. For example, if neuron N_1 gets the highest probabilities, the action setting reserved memory as 0.5 GB will be chosen, and if neuron N_2 gets the highest probabilities, the action setting reserved memory as 1 GB will be chosen. And so on, for each of the neurons.

5 Adaptive Scheme

Since MEER+ and Deep-MEER has different superiority and condition for execution, they are suitable for different cluster stages and complementary to each other. We will describe how to choose strategy adaptively during cluster life cycle in this section (Fig. 9).

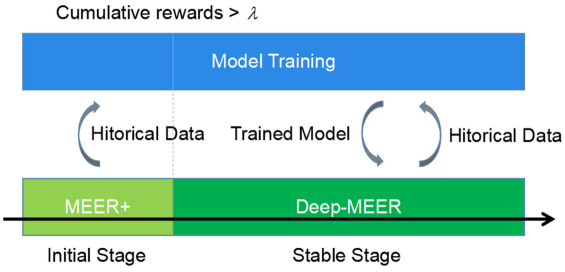

Fig. 9. Transition of strategy used during cluster life cycle.

Corresponding to estimation scheme, a cluster is divided into two stages:

Initial Stage. A cluster begins with initial stage. There is no prior experience when the cluster is just starting to work so MEER+ is adopted, sacrificing application performance several times to conduct the estimation. In the meantime, model training begins. A large amount of historical application execution data is produced in this stage, which is source material for model training.

Stable Stage. Stable Stage is the time when applications always perform well. The goal of reinforcement learning model is to maximize cumulative rewards. Once the cumulative rewards reach a threshold λ, we turn to adopting Deep-MEER and the cluster transforms into a stable stage. Threshold λ changes with application but it is easy to identify because it is the value cumulative rewards converging to. Of course it can be set a little smaller for fear of over fitting. Data from this stage can still be used to further train the model.

6 Evaluation Results

In order to evaluate our proposed schemes, we measure the performance of MEER+ and Deep-MEER from four aspects. (1) Estimation accuracy. Both schemes show nearer results to performance inflection point. (2) Generalization ability. The well-trained actor-critic model performs well even on workloads outside the training dataset. (3) Memory utilization. Idle memory is reduced by both of the proposed schemes to improve resource efficiency. (4) Application performance. Applications never suffer from performance plummeting under the estimation of Deep-MEER. In our experiments, we compare MEER+ and Deep-MEER with the state-of-the-art strategy MEER.

6.1 Experimental Setup

Through experiment in Sect. 1, we have proved the existence of performance inflection point in even 1-server-cluster, so we continue to conduct all the tests on this 1-server-cluster. The virtual machine is equipped with 1-core CPU, 12 GB memory and 200 GB virtual hard disks. We use Ubuntu 18.04 with Hadoop Yarn 2.7.3 and Spark 2.4.4 deployed. The only host acts as management node, HDFS data node and computing node simultaneously.

Table 2. Input size configuration of each workload.

Workload	Input size
Page Rank	360 MB
Shortest Paths	1900 MB
Triangle Count	165 MB
SVD++	170 MB

We run four representative workloads coming from SparkBench, Spark's benchmark performance test project for evaluation. In order to simulate the characteristics of application execution in industry, we configure the input size of each workload as shown in Table 2.

Both histogram analysis model and actor-critic learning model are implemented with Python. Execution time of the applications is obtained from history server of Spark and memory footprints are recorded in the metrics system. In the implementation of MEER+, we set $M_f = 0.5$, which ensures that formula (4) or (5) is met, but is not too conservative to limit the search speed. While computing loss value of Deep-MEER, we define the loss function as:

$$\text{Loss} = \frac{1}{n}\sum_{n=1}^{N}(L + L_S), \tag{8}$$

where N is the number of state transitions. Both MEER and MEER+ are able to complete the estimation within five loops of iterations on all workloads. Accordingly, we set N as five.L and L_S are two typical loss functions. Specifically, L is defined as:

$$\text{L} = -\text{logP} \times \delta_t. \tag{9}$$

And L_S is defined as:

$$L_S = \begin{cases} 0.5x^2, |\delta_t| < 1 \\ |\delta_t| - 0.5, otherwise \end{cases}. \tag{10}$$

To demonstrate the generalization ability of Deep-MEER, we run Page Rank and Triangle Count in initial stage to train the model and use the model to estimate on Shortest Paths and SVD++ in stable stage. What's more, we use MEER+ to estimate on Page Rank and Triangle Count independently for comparison.

For outstanding the advantages of our work, we take MEER as a contrast, showing and analyzing the experimental results from four aspects: estimation accuracy, generalization ability, memory utilization and application performance. Let's make agreement on some evaluation metrics before the analysis: (1) To describe estimation accuracy, we take ratio of differences between estimation result and the ground truth $\frac{M_n - M}{M}$ as a metric, which will be called relative error later for simplicity. Smaller error indicates higher accuracy. The generalization ability of model will be reflected by the relative error of the model's estimation results on the workloads outside the training dataset. (2) To describe memory utilization, we take the ratio of memory usage and estimated memory reservation $\frac{memory\_footprint}{M}$ as a metric. The higher the ratio, the higher the memory utilization. (3) To describe application performance, we use application execution time T as a metric, and there is no doubt that shorter execution time means better application performance.

6.2 Estimation Accuracy

An important goal of MEER+ and Deep-MEER is to find exactly the performance inflection point. Figure 10 shows the relative errors of final result of each strategy on the four typical workloads. Our results confirm that MEER+ has the best estimation

ability, showing the best accuracy in three-quarters of the tested workloads. Although it doesn't perform well in workload Shortest Path, the difference between it and others is next to nothing with only 0.17. This situation is reasonable if the step size in Stage 3 is not accurate enough. Ordering reserved memory from largest to smallest, the theoretical inflection point of performance is the point where performance fluctuates greatly, not necessarily the first point that satisfies the termination condition or not. However, Stage 3 of MEER+ takes it as a signal to determine the ultimate estimated optimal reservation. This is the reason why if the step size is set too large or too small, it may cause some slight deviation from the actual value. As for Deep-MEER, it outstands in Shortest Path and SVD++. What's more, it performs far better than MEER in Page Rank. Despite the result of Deep-MEER is not satisfying in Triangle Count, the error is on a par with that of MEER's worst result in Page Rank. Since reinforcement learning learns memory occupation rule conforming to most workloads rather than a specific workload, errors are inevitable.

To have an overall evaluation of each scheme's estimative power, we compute average relative errors of all workloads for each strategy and show them in Fig. 11. It is observed

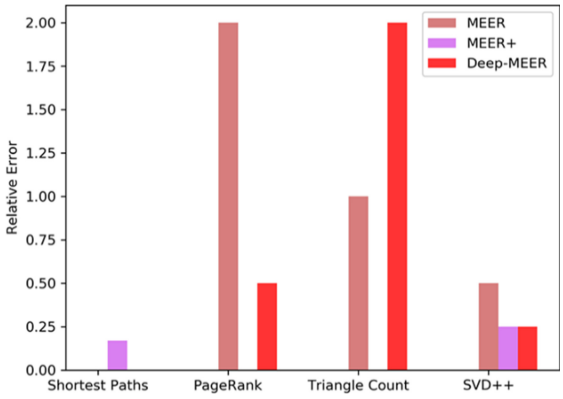

Fig. 10. Relative error of MEER, MEER+ and Deep-MEER on four benchmark workloads.

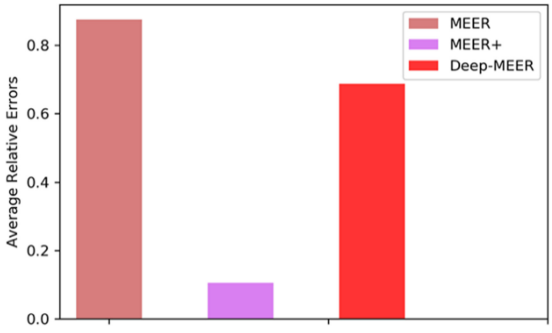

Fig. 11. Average relative error of MEER, MEER+ and Deep-MEER on four benchmark workloads.

that MEER+ shows the best preciseness with average relative error of 0.105. Deep-MEER comes after with average relative error of 0.688. They both perform better than MEER whose average relative error is 0.875. We can draw a conclusion that MEER+ and Deep-MEER reduce the relative error by 88% and 20% respectively.

6.3 Generalization Ability

The experimental results in Fig. 10 also indicates that reinforcement learning model based Deep-MEER has the advantages of excellent generalization ability. When adopting MEER or MEER+, the workload's estimated result is unique to itself, being not able to be applied to others. This is because the memory reservation is estimated only based on characteristics of memory usage of the current workload without any knowledge about that of other workloads. Whenever a new application is submitted, a new round of estimation will be activated. However, Deep-MEER learns the general rule of memory usage in multiple workloads, so the estimation model is applicable for any workloads. In our experiment, the actor-critic model is trained with data from workload Page Rank and Triangle Count but when we use the trained model to estimate on Shortest Paths and SVD++, the estimation result is still good with little relative error of 0 and 0.25. It is inspiring to know that once the model is trained, it is applicable for various workloads, which means for a cluster that runs millions of applications, the cost of training model can be ignored. Data centers may just need to pre-train a model using test data in trial operations of a small number of workloads, then the model can be applied directly in formal operations. Real-time data from new applications that use Deep-MEER to determine reserved memory size can also be utilized to train and perfect the model, making the model more inclusive.

6.4 Memory Utilization

The purpose of reducing memory reservation as much as possible is to improve memory utilization and save resource without secure impact on application performance. We calculate the memory utilization every second in application execution. Taking Page Rank as an example, we draw a diagram of the memory utilization changing with execution time. As shown in Fig. 12, the memory utilization of MEER+ and Deep-MEER is always greater than MEER, and their average utilization is about 21%, twice that of MEER. The peak memory utilization of MEER+ and Deep-MEER is 47% and 49% respectively, while the peak of MEER is only 23%. This is as expected, since MEER stops iterative search as soon as it meets the termination condition, which is rough and conservative, leading that the estimation result is often larger than the performance inflection point. However, the actual execution of the application does not need so much memory resource, so the memory utilization is not high. MEER+ effectively reduces the redundant memory reservation in approximation stage, and Deep-MEER improves the resource efficiency by learning the rule of memory occupation to make a closer estimation to the optimal value.

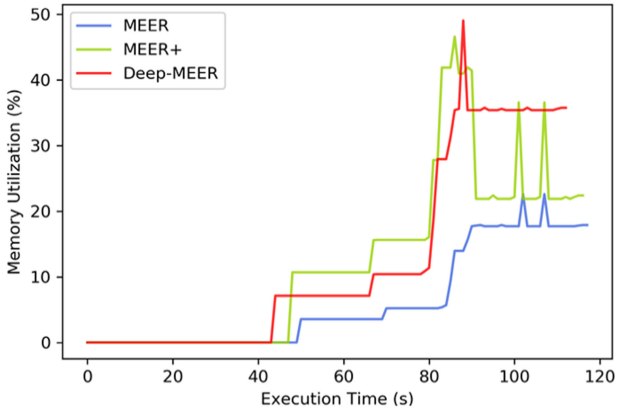

Fig. 12. Variants of memory utilization during the execution of Page Rank.

6.5 Application Performance

Thrashing-avoiding is another strength of Deep-MEER to ensure the stability of application performance, which means it is more user-friendly and more suitable for enterprises. We take an overall view of applications' performance in each recurring submission to illustrate how Deep-MEER protects applications from radical performance change.

Figure 13 is execution time in recurring executions of Page Rank and Shortest Paths. Both MEER and MEER+ experience a cliff of performance degradation in the process of estimation. This is caused by termination conditions, to meet which the application may go through a low performance run. MEER+ may experience more times of low performance for the stage of approximation may approach the optimal under inadequate reserved memory. In terms of Deep-MEER, its model always gets low rewards while

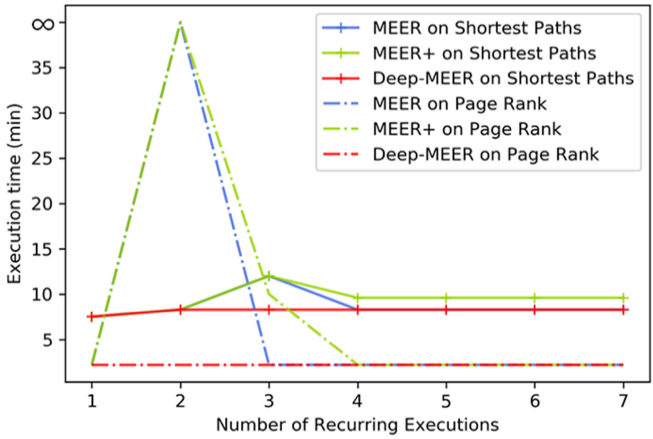

Fig. 13. Variations of execution time during recurring executions using MEER, MEER+ and Deep-MEER on benchmark workloads (Dotted lines refer to Shortest Paths and solid lines refer to Page Rank).

setting reserved memory size less than the performance inflection point in the process of training. Drawing a lesson from history, it mostly chooses memory settings that larger than inflection point to gain high rewards. This can explain why using Deep-MEER, workloads can always maintain satisfying performance.

7 Conclusion

In this paper, we present MEER+ and Deep-MEER, which are derived from MEER. MEER+ and Deep-MEER are designed to assist resource manager in estimating optimal memory reservation for long-lived containers accurately and effectively in data centers. By adding a process of approximation into MEER, MEER+ reverses research direction to rectify the estimated result. By combining fashionable reinforcement learning technique to explore the pattern of memory reservation, Deep-MEER exploits the potential of abundant historical data and prevents applications from being affected by termination conditions. The two schemes can be adopted adaptively in cluster life cycle to achieve trade-off between execution time and resource utilization. Both MEER+ and Deep-MEER have shown promising results in experiments.

Acknowledgments. This work is supported by The National Key Research and Development Program of China (2019YFB1804502), Key-Area Research and Development Program of Guangdong Province (Grant No.2019B010107001), and National Natural Science Foundation of China (Grant No.61832020, 61702569).

References

1. Zhang, W., Wang, L., Cheng, Y.: Performance optimization of Lustre file system based on reinforcement learning. J. Comput. Res. Dev. **56**(7), 1578–1586 (2019)
2. Zhao, T., Dong, S., March, V., et al.: Predicting the parallel file system performance via machine learning. J. Comput. Res. Dev. **48**(7), 1202–1215 (2011)
3. Boutin, E., Ekanayake, J., Lin, W., et al.: Apollo: scalable and coordinated scheduling for cloud-scale computing. In: 11th Symposium on Operating Systems Design and Implementation (OSDI), Broomfield, CO, pp. 285–300 (2014)
4. Zhu, Y., Liu, J., Guo, M., et al.: Bestconfig: tapping the performance potential of systems via automatic configuration tuning. In: 2017 Symposium on Cloud Computing (SoCC), Santa Clara, California, pp. 338–350 (2017)
5. Vavilapalli, V.K., Murthy, A.C., Dougla, C., et al.: Apache Hadoop YARN: yet another resource negotiator. In: 4th Annual Symposium on Cloud Computing (SoCC), Santa Clara, California, no. 5, pp. 1–16 (2013)
6. Hindman, B., Konwinski, A., Zaharia, M., et al.: Mesos: a platform for fine-grained resource sharing in the data center. In: 8th conference on Networked Systems Design and Implementation (NSDI), Boston, MA, pp. 295–308 (2011)
7. Schwarzkopf, M., Konwinski, A., Abd-El-Malek, M., et al.: Omega: flexible, scalable schedulers for large compute clusters. In: 8th ACM European Conference on Computer Systems (EuroSys), Prague, Czech Republic, pp. 351–364 (2013)
8. Verma, A., Pedrosa, L., Korupolu, M., et al.: Large-scale cluster management at Google with Borg. In: 10th European Conference on Computer Systems (EuroSys), Bordeaux, France, no. 18, pp. 1–17 (2015)

9. Burns, B., Grant, B., Oppenheimer, D., et al.: Borg, Omega, and Kubernetes. In: Communications of the ACM, New York, USA, vol. 59, no. 5, pp. 50–57 (2016)
10. Xu, G., Xu, C.: MEER: online estimation of optimal memory reservations for long lived containers in in-memory cluster computing. In: 39th IEEE International Conference on Distributed Computing Systems (ICDCS), Dallas, TX, USA, pp. 23–34 (2019)
11. Garefalakis, P., Karanasos, K., Pietzuch, P., et al.: Medea: scheduling of long running applications in shared production clusters. In: 13th EuroSys Conference, Porto, Portugal, no. 4, pp. 1–13 (2018)
12. Chen, H., Jiang, G., Zhang, H., et al.: Boosting the performance of computing systems through adaptive configuration tuning. In: 2009 ACM Symposium on Applied Computing (SAC), Honolulu, Hawaii, pp. 1045–1049 (2009)
13. Abadi, M., Barham, P., Chen, J., et al.: Tensorflow: a system for large-scale machine learning. In: 12th conference on Operating Systems Design and Implementation (OSDI), Savannah, GA, USA, pp. 265–283 (2016)
14. Meng, X., Bradley, J., Yavuz, B., et al.: Mllib: Machine learning in apache spark. J. Mach. Learn. Res. **17**(1), 1235–1241 (2016)
15. Zaharia, M., Chowdhury, M., Das, T., et al.: Resilient distributed datasets: a faulttolerant abstraction for in-memory cluster computing. In: 9th Conference on Networked Systems Design and Implementation (NSDI), San Jose, CA, p. 2 (2012)
16. Apache flink. http://flink.apache.org. Accessed 30 Mar 2020
17. Toshniwal, A., Taneja, S., Shukla, A., et al.: Storm@twitter. In: 2014 ACM SIGMOD International Conference on Management of Data, Snowbird, Utah, USA, pp. 147–156 (2014)
18. Zaharia, M., Das, T., Li, H., et al.: Discretized streams: fault-tolerant streaming computation at scale. In: 24th ACM Symposium on Operating Systems Principles (SOSP), Farminton, Pennsylvania, pp. 423–438 (2013)
19. Armbrust, M., Xin, R.S., Lian, C., et al.: Spark SQL: relational data processing in spark. In: 2015 ACM SIGMOD International Conference on Management of Data, Melbourne, Victoria, Australia, pp. 1383–1394 (2015)
20. Kornacker, M., Behm, A., Bittorf, V., et al.: Impala: a modern, open-source SQL engine for hadoop. In: 7th Biennial Conference on Innovative Data Systems Research (CIDR) (2015)
21. Saha, B., Shah, H., Seth, S., et al.: Apache Tez: a unifying framework for modeling and building data processing applications. In: 2015 ACM SIGMOD International Conference on Management of Data, Melbourne, Victoria, Australia, pp. 1357–1369 (2015)
22. Gonzalez, J.E., Xin, R.S., Dave, A., et al.: Graphx: graph processing in a distributed dataflow framework. In: 11th Conference on Operating Systems Design and Implementation (OSDI), Broomfield, CO, pp. 599–613 (2014)
23. Low, Y., Bickson, D., Gonzalez, J., et al.: Distributed graphlab: a framework for machine learning and data mining in the cloud. VLDB Endow. **5**(8), 716–727 (2012)
24. Malewicz, G., Austern, M.H., Bik, A.J.C., et al.: Pregel: a system for large-scale graph processing. In: 2010 ACM SIGMOD International Conference on Management of Data, Indianapolis, Indiana, USA, pp. 135–146 (2010)
25. Iorgulescu, C., Dinu, F., Raza, A., et al.: Don't cry over spilled records: memory elasticity of data-parallel applications and its application to cluster scheduling. In: Annual Technical Conference (ATC), pp. 97–109 (2017)
26. Herodotou, H., Dong, F., Babu, S.: No one (cluster) size fits all: automatic cluster sizing for data-intensive analytics. In: 2nd ACM Symposium on Cloud Computing (SoCC), Cascais, Portugal, no. 18, pp. 1–14 (2011)
27. Alipourfard, O., Liu, H.H., Chen, J., et al.: Cherrypick: adaptively unearthing the best cloud configurations for big data analytics. In: 14th Symposium on Networked Systems Design and Implementation (NSDI), Boston, MA, pp. 469–482 (2017)

28. Huynh, N.V., Nguyen, D.N., Dutkiewicz, E.: Optimal and fast real-time resource slicing with deep dueling neural networks. IEEE J. Sel. Areas Commun. **37**(6), 1455–1470 (2019)
29. Xu, G., Xu, C.: Prometheus: online estimation of optimal memory demands for workers in in-memory distributed computation. In: The ACM Symposium on Cloud Computing (SoCC), Santa Clara, California, p. 655 (2017)
30. Chen, W., Pi, A., Wang, S., et al.: Pufferfish: container-driven elastic memory management for data-intensive applications. In: the ACM Symposium on Cloud Computing (SoCC), Santa Cruz, CA, USA, pp. 259–271 (2019)
31. Klimovic, A., Litz, H., Kozyrakis, C.: Selecta: heterogeneous cloud storage configuration for data analytics. In: 2018 USENIX Annual Technical Conference (ATC), Boston, MA, USA, pp. 759–773 (2018)
32. Liu, L., Xu, H.: Elasecutor: elastic executor scheduling in data analytics systems. In: The ACM Symposium on Cloud Computing (SoCC), Carlsbad, CA, USA, pp. 107–120 (2018)
33. Peng, G., Wang, H., Dong, J., et al.: Knowledge-based resource allocation for collaborative simulation development in a multi-tenant cloud computing environment. IEEE Trans. Serv. Comput. (TSC) **11**(2), 306–317 (2018)
34. Erradi, A., Iqbal, W., Mahmood, A., et al.: Web application resource requirements estimation based on the workload latent features. IEEE Trans. Serv. Comput. (TSC), 1(2019)
35. Kholidy, H.A.: An intelligent swarm based prediction approach for predicting clou computing user resource needs. Comput. Commun. (CC) **151**, 133–144 (2020)