DeepCAT: A Cost-Efficient Online Configuration Auto-Tuning Approach for Big Data Frameworks

Hui Dou
douhui@ahu.edu.cn
Anhui University
Hefei, Anhui, China

Yilun Wang
wangyilun@stu.ahu.edu.cn
Anhui University
Hefei, Anhui, China

Yiwen Zhang
zhangyiwen@ahu.edu.cn
Anhui University
Hefei, Anhui, China

Pengfei Chen
chenpf7@mail.sysu.edu.cn
Sun Yat-sen University
Guangzhou, Guangdong, China

ABSTRACT

To support different application scenarios, big data frameworks usually provide a large number of performance-related configuration parameters. Online auto-tuning these parameters based on deep reinforcement learning to achieve a better performance has shown their advantages over search-based and machine learning-based approaches. Unfortunately, the time consumption during the online tuning phase of conventional DRL-based methods is still heavy, especially for big data applications. Therefore, in this paper, we propose DeepCAT, a cost-efficient deep reinforcement learning-based approach to achieve online configuration auto-tuning for big data frameworks. To reduce the total online tuning cost: 1) DeepCAT utilizes the TD3 algorithm instead of DDPG to alleviate value overestimation; 2) DeepCAT modifies the conventional experience replay to fully utilize the rare but valuable transitions via a novel reward-driven prioritized experience replay mechanism; 3) DeepCAT designs a Twin-Q Optimizer to estimate the execution time of each action without the costly configuration evaluation and optimize the sub-optimal ones to achieve a low-cost exploration-exploitation trade off. Experimental results based on a local 3-node Spark cluster and HiBench benchmark applications show that DeepCAT is able to speed up the best execution time by a factor of 1.45× and 1.65× on average respectively over CDBTune and OtterTune, while consuming up to 50.08% and 53.39% less total tuning time.

KEYWORDS

Performance Optimization; Online Configuration Tuning; Big Data Framework; Deep Reinforcement Learning

ACM Reference Format:
Hui Dou, Yilun Wang, Yiwen Zhang, and Pengfei Chen. 2022. DeepCAT: A Cost-Efficient Online Configuration Auto-Tuning Approach for Big Data Frameworks. In *51st International Conference on Parallel Processing (ICPP '22), August 29-September 1, 2022, Bordeaux, France.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3545008.3545018

ICPP '22, August 29-September 1, 2022, Bordeaux, France

ACM ISBN 978-1-4503-9733-9/22/08…$15.00
https://doi.org/10.1145/3545008.3545018

1 INTRODUCTION

In order to adapt to various application scenarios, big data frameworks such as HDFS, Spark and Flink usually provide a large number of performance-related parameters for developers. Inappropriate configuration of these parameters may cause performance degradation [13] and even non-functional faults[14]. Since manually configuration tuning is both time-consuming and labor-intensive, how to automatically tune configuration parameters for big data frameworks have recently been a hot topic[2, 17, 22, 29, 32] in academia as well as industry. However, configuration tuning is not an once for all job because the performance of a big data framework under the same configuration is highly related to the workload characteristics (e.g., workload type and input data size) and its underlying hardware infrastructures[17, 24], which may frequently change with time in practice. Hence, there is an urgent requirement for an online configuration auto-tuning approach for big data frameworks.

Broadly speaking, there are mainly three challenges to auto-tune configuration parameters in a short time: **1)Complex relationship between parameters and performance:** Considering the complex implementation of big data frameworks, the relationship between performance and configuration parameters is very difficult to figure out. Besides, the relationship is highly related to the time-varying workload characteristics and hardware environment, which further increases its complexity. **2)Heavy configuration evaluation cost:** Due to the lack of reliable performance simulators, the time consumed on evaluating the performance of the big data framework under a certain configuration is typically long. We usually have to execute a benchmark application on a real big data cluster for several times, which will take several to tens of minutes for each evaluation. **3)High dimensional configuration space:** Each big data framework owns a large number of performance-related parameters to tune. These parameters can be numeric or categorical and together construct a high dimensional candidate configuration space. This space will further expand when we try to optimize a big data pipeline composed of several different frameworks.

Recently, deep reinforcement learning (DRL)-based methods [10, 18, 30] have shown their advantages in addressing above challenges over the search-based[16, 32] and machine learning-based[3, 26, 29]

approaches. Specifically, search-based approaches such as BestConfig [32] explore the candidate configuration space according to a certain heuristic in an online manner. Since they cannot effectively exploit historical experiences, these approaches usually need a large number of time-consuming configuration evaluation and restart from scratch whenever a new tuning request comes. On the other hand, machine learning-based approaches such as OtterTune[26] first offline model the relationship between parameters and performance and then execute online configuration tuning based on this learned model. However, these approaches need a large number of high-quality samples to train an accurate performance prediction model and what is worse, model re-training is usually necessary whenever workload or hardware change happens due to the poor adaptability of conventional machine learning algorithms. Different from above methods, DRL-based approaches such as CDBTune[30] first adopt an efficient trial-and-error strategy to offline train a DRL model and then fine-tune this standard model with several sequential online tuning steps to adapt to the real scenario. As a result, these approaches can recommend a good configuration for each newly arrived tuning request with only several times of the costly configuration evaluation.

Unfortunately, although DRL-based approaches are able to avoid the expensive offline model re-training due to their good adaptability, we insist that under big data application scenarios, the time consumed by their online tuning steps should not be ignored. For instance, in our experiments conducted on a local Spark cluster with the TeraSort benchmark application in HiBench and a modest input data size (3.2GB), the configuration evaluation time during the 5 online tuning steps of CDBTune vary from 37.5 seconds to as many as 109 seconds, leading to a remarkable total time consumption for an online configuration auto-tuning approach. In fact, there are two major limitations of prior DRL-based approaches when applied to online auto-tune big data frameworks:

Lack of an effective experience replay mechanism. Challenges discussed above means that for a big data framework, the close-to-optimal configurations are far fewer than sub-optimal ones and scattered throughout the high-dimensional configuration space. In order to fully utilize these sparse but valuable experiences, it is necessary to replay corresponding transitions. However, the conventional experience replay mechanism is not able to recognize the important experiences out since it samples transitions from the memory pool in a totally random way. Although a prioritized experience replay mechanism (PER)[25] is recently proposed to enhance DRL in scenarios with sparse rewards, unfortunately, PER aims to gain enough information from environment, which is infeasible and unnecessary for an online configuration auto-tuning problem. The lack of an effective experience replay mechanism results in a slow convergence rate of the offline training phase.

Neglecting the cost of each online tuning step. When a new configuration tuning request is arrived, DRL-based approaches need to fine-tune the offline trained model with several sequential online tuning steps to adapt to the real user environment. Each step will launch a costly configuration evaluation task and forms a remarkable cumulative total online tuning time. Nevertheless, prior DRL-based approaches only target on how to eventually find a good configuration, while totally overlooking the time consumption of each online tuning step. In addition to the quality of recommended configuration, an online configuration auto-tuning approach should pay more attention to its total tuning cost.

Towards these limitations of prior DRL-based methods, we propose a cost-efficient deep reinforcement learning-based online configuration auto-tuning approach named DeepCAT for big data frameworks in this paper. Overall, DeepCAT is able to find a better configuration consuming less total tuning time with the following two novel techniques: 1)**Reward-driven prioritized experience replay mechanism.** Instead of the conventional deep deterministic policy gradient algorithm (DDPG)[19], DeepCAT utilizes the twin delayed deep deterministic policy gradient (TD3)[9] since it can mitigate the value overestimation problem. To address the challenge of sparse close-to-optimal configurations of a big data framework, we design a reward-driven PER mechanism named RDPER in DeepCAT. The core insight is that for online configuration auto-tuning, we should pay more attention directly to the valuable high-reward experiences rather than high TD (temporal difference)-error transitions like PER did. With the aid of two memory pools, RDPER is able to guarantee the proportion of high-reward transitions in the total replayed samples so that DeepCAT can fully utilize these valuable experiences. 2) **Twin-Q Optimizer.** In order to reduce the total time consumed during the online tuning steps, we also propose a Twin-Q Optimizer algorithm in DeepCAT. Different from all prior DRL-based approaches, we construct a novel indicator based on the twin critic networks in TD3 to recognize the sub-optimal actions and avoid the corresponding costly configuration evaluation. These identified sub-optimal actions then will be optimized with a Gaussian noise and finally achieve a low-cost exploration-exploitation trade off for online tuning.

To evaluate the effectiveness and efficiency of DeepCAT when applied into online configuration auto-tuning for big data frameworks, we conduct extensive experiments on a local 3-node Spark cluster with 12 different workload-input pairs from HiBench. Experimental results show that DeepCAT is able to speed up the best execution time by a factor of 1.45× and 1.65× on average respectively over CDBTune and OtterTune, while consuming 24.64% and 39.71% on average and up to 50.08% and 53.39% less total tuning time. In addition, DeepCAT also has a good adaptability to the time-varying environment of big data frameworks.

2 SYSTEM OVERVIEW

We propose a DRL-based approach named DeepCAT to achieve cost-efficient online configuration auto-tuning for big data frameworks. Figure 1 shows the architecture of DeepCAT, which is composed of an offline training stage and an online tuning stage. The offline training stage is responsible for training a DRL model based on the interactions with the standard environment. When an online configuration tuning request is arrived, the online tuning stage is responsible for fine-tuning the offline model via a certain number of online tuning steps to recommend a satisfying configuration for users. We briefly introduce each stage here and the details can be found in Section 3.

Offline Training Stage. In order to train an effective DRL model for online tuning, DeepCAT chooses to use the TD3 algorithm[9] to address the value overestimation problem of DDPG[19]. For big

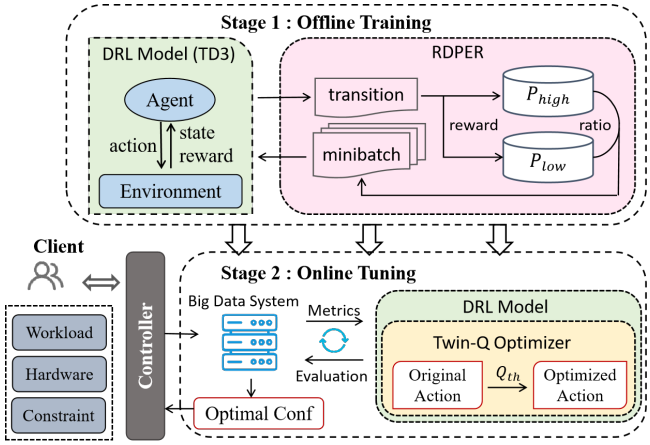

Figure 1: System architecture of DeepCAT

data applications, close-to-optimal configurations are far fewer than sub-optimal ones. Therefore, DeepCAT utilizes a reward-driven prioritized experience relay mechanism named RDPER to take advantages of the sparse high-reward transitions. RDPER divides the original samples separately into a high-reward and low-reward memory pool and guarantees the ratio of high-reward transitions to the total replayed batch samples. Note that the DRL model is only offline trained once and then can be used to online auto-tune configurations for various online tuning requests from users.

Online Tuning Stage. When a new configuration tuning request is arrived, the online tuning stage will start to fine-tuning the offline DRL model for the real user environment (workload, hardware) within the specified tuning cost constraint. For each step, DeepCAT use the actor network in TD3 to output an action according to current system state. In order to reduce the tuning cost, DeepCAT utilize the Twin-Q Optimizer to estimate the real execution time of this recommended configuration and optimize the sub-optimal ones. After that, the optimized configuration will be evaluated on the target big data cluster to obtain the corresponding performance. When the number of tuning steps reaches the constraint or the total tuning time exceeds the budget, DeepCAT will terminate and report the best configuration ever found to users.

3 DESIGN AND IMPLEMENTATION OF DEEPCAT

3.1 DRL Formulation

In order to achieve cost-efficient online configuration auto-tuning for big data frameworks, we proposed DeepCAT based on deep reinforcement learning. For the first step, we translate the configuration auto-tuning problem into a DRL formulation. Considering the limit of space, here we present three core components in DRL:

State: State s_t is an internal indicator of the target environment, which refers to the current environment status. For a big data cluster, once DeepCAT recommends a configuration and begins to evaluate it, we choose to use the `uptime` command to obtain the load averages of each server to represent the current state.

Action: Different from DQN[21], we try to learn deterministic policies where each action a_t can directly represent the configuration of candidate parameters. To auto-tune parameters from the big data framework, DeepCAT performs corresponding actions under current states according to the latest policy. Noting that each dimension in a_t is normalized to [0,1] to tackle with the different categories (numerical, categorical, etc.) as well as various value scales of different parameters.

Immediate Reward: Reward directs the agent to learn and thus is vital in DRL. Since we aim to improve the performance of a big data framework, the reward r_t should be determined based on the performance $perf_t$ after an action a_t is executed. In this work, we design an effective immediate reward function to guide the agent to find a desired good configuration:

$$r_t = \frac{perf_e - perf_t}{perf_e} \tag{1}$$

where $perf_e$ is the expected performance of the target system. According to the performance improvement achieved by prior studies, we set $perf_e$ to be a speedup with respect to the default execution time. It is worth noting that for configuration auto-tuning problem, the agent is able to conduct configuration evaluation to receive a feedback for each action with this reward function. These immediate rewards can speed up the convergence rate of offline training. Besides, compared with prior studies[10, 18, 30] which target on an eventual optimal performance, our immediate reward mechanism is consistent with the tuning goal and can drive the agent to pursue maximizing the performance for every single action, which is helpful to reduce the total time consumption of online tuning steps.

3.2 DRL for tuning

Since there are a considerable number of configuration parameters from a big data pipeline, finding the optimal configuration in this high-dimensional continuous candidate space is NP-hard[30]. Therefore, we give a short introduction of DDPG and TD3 algorithm and explain their limitations in achieving cost-efficient online configuration auto-tuning.

Deep Deterministic Policy Gradient[19]: DDPG is a policy-based DRL algorithm which is able to perform well in the high-dimensional continuous action space. It combines the Actor-Critic method with the successful experience of DQN and the execution process is as following:

(1) Actor selects an action and critic calculates the corresponding Q-value of this action as a feedback, where the Q function $Q^{\mu}(s,a)$ is defined by Bellman Equation[27]:

$$Q^{\mu}(s,a) = \mathbb{E}_{r_t, s_{t+1}}\left[r(s_t,a_t) + \gamma Q^{\mu}(s_{t+1}, \mu(s_{t+1}))\right] \tag{2}$$

where μ is policy, s_{t+1} is the next state, $r(s_t, a_t)$ is the reward function and γ is a discount factor denotes the correlation between future and present rewards.

(2) Temporary difference (TD) algorithm is used to train the critic network, parameters are updated with gradient descent to minimize the loss function $L\left(\theta^{Q}\right)$:

$$L\left(\theta^{Q}\right)=\mathbb{E}\left[\left(Q\left(s,a\mid\theta^{Q}\right)-y\right)^{2}\right] \tag{3}$$

where

$$y=r\left(s_t,a_t\right)+\gamma Q^{\mu}\left(s_{t+1},\mu\left(s_{t+1}\right)\mid\theta^{Q}\right)$$

(3) According to the feedback of critic, actor updates the policy with the policy gradient:

$$\nabla_{\theta^{\mu}}J\approx\mathbb{E}\left[\nabla_{a}Q\left(s,a\mid\theta^{Q}\right)\mid_{s=s_t,a=\mu(s_t)}\nabla_{\theta^{\mu}}\mu\left(s\mid\theta^{\mu}\right)\mid_{s=s_t}\right] \tag{4}$$

DDPG extends DQN to a high-dimensional continuous space, and has been recently applied to auto-tune configuration parameters for databases. However, the learned value function may significantly overestimate the Q-value, which leads to a poor trained policy network. As a result, the agent will fail to recommend good actions and lead to time-consuming configuration evaluation during the online tuning stage.

Twin Delayed Deep Deterministic Policy Gradient[9] : TD3 algorithm aims to address the value overestimation problem of DDPG. To this end, TD3 adopts two critic networks and updates them separately. To offset the overestimation in the Q function, the smaller output of two critics will be selected to estimate the target Q-value:

$$y_i=r\left(s_t,a_t\right)+\gamma\min_{i=1,2}Q_i^{\mu}\left(s_{t+1},\mu\left(s_{t+1}\right)\mid\theta^{Q_i}\right)$$

Each critic network is independently updated similar to DDPG:

$$minL\left(\theta^{Q_1}\right)=\mathbb{E}\left[\left(Q_1\left(s,a\mid\theta^{Q_1}\right)-y_1\right)^{2}\right]$$

$$minL\left(\theta^{Q_2}\right)=\mathbb{E}\left[\left(Q_2\left(s,a\mid\theta^{Q_2}\right)-y_2\right)^{2}\right]$$

In addition, TD3 also performs target policy smoothing as well as delaying policy update to provide a more stable training process than DDPG. Hence, we choose to utilize TD3 to implement DeepCAT. Unfortunately, the sparse high-reward transitions and the time-consuming online tuning steps needed to adapt to the real user environment still prevent DeepCAT from achieving a cost-efficient online configuration auto-tuning for big data frameworks.

3.3 RDPER: Reward-driven Prioritized Experience Replay

Figure 2 illustrates the cumulative probability distribution of 200 random generated configurations for TeraSort according to their relative performance to the found optimal configuration. We can find that although it is easy to find a better-than-default configuration, the close-to-optimal configurations are far fewer than sub-optimal ones and scattered throughout the high-dimensional configuration space. In order to take fully advantages of the valuable but sparse positive rewards when auto-tuning configuration for a big data pipeline, a straightforward method is to replay corresponding transitions according to a certain policy. Since conventional experience replay mechanism utilized by the off-policy RL sample transitions from the memory pool in a totally random way, it fails to recognize the important transitions. Therefore, prioritized experience replay (PER)[25] is currently widely used to enhance DRL in scenarios with sparse rewards. Specifically, PER utilizes the temporal difference (TD) error to represent the priority of each transition

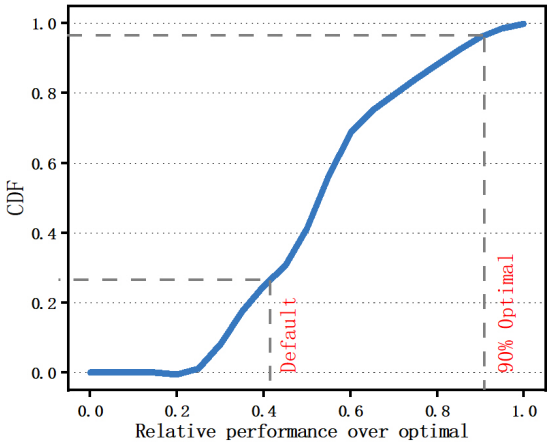

Figure 2: The CDF of 200 random generated configurations.

and replays the high-priority transitions to gain more environment information.

However, the objective of online configuration auto-tuning is to find the best configuration in a short time, trying to collect enough information from environment like PER is infeasible and unnecessary. Therefore, instead of the transitions with a high TD error, we should pay more attention to the sparse high-performance actions. In fact, according to Eq. (4), the policy improvement can be decomposed into the gradient of the Q with respect to actions and the gradient of the actions with respect to the policy parameters based on the chain rule. Therefore, transitions with a larger estimated Q-value can obtain greater policy update information. Since high-reward transitions obtain a larger estimated Q-value according to Eq. (2), it is believed that replaying these transitions more frequently will speed up the convergence rate of policy network training and do good to policy improvement. To this end, we design a reward-driven prioritized experience replay mechanism named RDPER for DeepCAT. The core insight of RDPER is that we divided the transitions into two categories ([*high reward*, *low reward*]) according to the relationship of size between their rewards and a pre-specified threshold R_{th}. Specifically, DeepCAT utilizes two memory pools denoted as P_{high} and P_{low} to separately store these transitions so as to guarantee the ratio of high-reward transitions to replayed transition samples. When the learning process starts, transitions with an equal or higher reward than R_{th} will be stored in P_{high} and others will be stored in P_{low}. If we denote the batch size as m and the ratio of high-reward transitions as β, βm transitions will be sampled from P_{high} and $(1-\beta)m$ transitions from P_{low} to train the networks once DeepCAT obtains enough transitions. How to determine the value of hyper-parameter β will be discussed in Section 5.4.1.

Through this reward-driven PER mechanism, DeepCAT is able to make a good use of the valuable but sparse high-reward experiences and thus speed up the convergence rate of offline training as well as improve configuration performance. The effectiveness of this mechanism for training TD3 in DeepCAT will also be evaluated in Section 5.1.1.

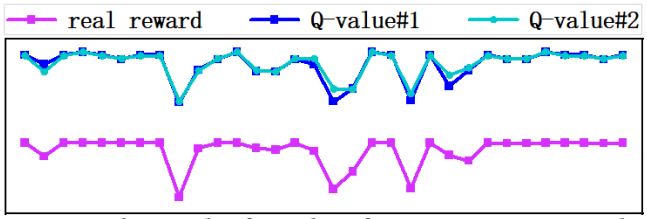

Figure 3: The trends of Q-values from twin critic networks of TD3 and the real reward.

3.4 Twin-Q Optimizer

When a configuration auto-tuning request is arrived, DeepCAT need to fine-tune the offline trained model to adapt to the real user environment. This process usually costs several online tuning steps before it can recommend a satisfying configuration for users. Unfortunately, even with the improvements described above, fine-tuning will still have to evaluate the time-consuming sub-optimal configurations when the offline trained model is utilized in a new scenario, resulting in an non-ignorable total online tuning cost.

To address this problem, we propose the Twin-Q Optimizer algorithm in DeepCAT. The major idea of this algorithm is to estimate the real execution time of each online recommended action and replace the sub-optimal configurations with estimated promising ones inherit from themselves. Since conventional DRL totally neglects the cost of each online tuning step, it is challenging for our algorithm to obtain an estimation of current actions. However, for the configuration auto-tuning problem, we find that the Q-value of an action is a good indicator to its cost. In fact, RL utilizes a value function to estimate the quality of current situation based on the expected future rewards. If we use U_t to denote the total discounted reward and $Q(s_t, a_t)$ to denote the value function, we can obtain that:

$$U_t = R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \gamma^3 R_{t+3} + \cdots \tag{5}$$

$$Q(s_t, a_t) = \mathbb{E}[U_t \mid S_t = s_t, A_t = a_t] \tag{6}$$

As a result, we can use the output of the offline trained critic network to identify whether the current recommended action is sub-optimal. It is worth noting that prior DRL-based approaches totally discarded the utilization of the already trained critic network during their online tuning phase. In order to further improve the accuracy of configuration evaluation, we take the advantage of the twin critic networks in TD3 and use the smaller of the Q-values as the evaluation metric, which is inspired by the parameter update method of TD3. Figure 3 shows the smaller of the twin-Q indeed share a very similar trend with the real reward during the offline training stage. Experimental results in Section 5.1.2 verify the effectiveness of this Twin-Q indicator mechanism for online tuning.

For the identified sub-optimal configurations, we add a Gaussian noise $\epsilon \sim N(0, \sigma_\epsilon^2)$ to the original action and evaluate the newly generated action with above Twin-Q indicator mechanism. This exploration process will repeat until an close-to-optimal action is recommended. Note that since no actions are actually executed during this optimization procedure, this method is able to avoid the time-consuming execution of sub-optimal actions with only a slight cost. Therefore, the total time consumption of the whole fine-tuning phase can be obviously reduced.

Algorithm 1: Twin-Q Optimizer

Input:
state s, action a, Q-value threshold Q_{th}
Output:
an optimized action a*

```
1: while do
2:   q1 = Critic1(s, a) , q2 = Critic2(s, a)
3:   if min(q1, q2) ≥ Q_th then
4:     a* = a
5:     break
6:   else
7:     modify action a with Gaussian noise ε: a = a + ε
8:   end if
9: end while
```

Combing the above identification and replacement mechanisms together, we finally propose the Twin-Q Optimizer algorithm in DeepCAT. As described in Algorithm 1, for each recommended action a during the online tuning stage, we respectively utilize the two critic networks to obtain its Q-values q_1 and q_2 (line 2). Then we compare the smaller one with a pre-defined Q-value threshold Q_{th} to determine whether action a is sub-optimal. For the sub-optimal actions, we add a Gaussian noise ϵ to generate a new action and repeat this operation until the newly generated action is identified as close-to-optimal (line 3-8). It is worth noting that a larger Q_{th} increases the exploration around the sub-optimal space, while a lower Q_{th} tends to exploit the good configuration has been already found. How to determine it to achieve a low-cost exploration-exploitation trade off will be discussed in Section 5.4.2.

3.5 Advantages

Our method has the following advantages in online configuration auto-tuning for big data frameworks: 1)**High-dimensional configuration recommendation.** The deep neural networks in TD3 have a much better representation ability to learn the high dimensional configuration space than ML models and avoid the value overestimation problem of DDPG. 2)**Supporting scenarios with sparse high-reward transitions**. With the RDPER mechanism, DeepCAT makes a fully use of the rare high-reward transitions and speeds up the convergence rate of offline training as well as improves configuration performance. 3)**Less total online tuning cost.** With the Twin-Q Optimizer, DeepCAT is able to estimate the real execution time of each online recommended action and optimize the sub-optimal ones to achieve a low-cost exploration-exploitation trade off. 4)**Good Adaptability.** Unlike traditional ML methods which highly rely on the training data, DRL learns a policy to output a good configuration according to system state and thus has a good adaptability to the new environment. The RDPER and Twin-Q Optimizer techniques also help guarantee the effectiveness and efficiency when environment changes.

4 EXPERIMENTAL SETUPS

4.1 Experimental Platform

In our experiments, we use DeepCAT to online auto-tune configurations for a big data pipeline composed of HDFS, Yarn and Spark.

Table 1: Workload characteristics

Workload	Category	Input Datasets (D1, D2, D3)
WordCount (WC)	micro	3.2, 10, 20 (GB)
TeraSort (TS)	micro	3.2, 6, 10 (GB)
PageRank (PR)	websearch	0.5, 1, 1.6 (Million Pages)
KMeans (KM)	ML	20, 30, 40 (Million Points)

Table 2: Number of tuned parameters in the pipeline

Component of the pipeline	Number of parameters
Spark	20*
YARN	7
HDFS	5

*Including the Spark-YARN connector parameters

Our experimental platform is a local 3-node Spark cluster. All the servers are connected with a 1-Gigabit Ethernet network and each server is equipped with one Intel(R) Core(TM) i7-10700 2.9GHz with 16 physical cores, 16 GB DDR4 memory and 1TB HDD. The version of the frameworks is Apache Spark 2.2.2 and Hadoop 2.7.3, while the underlying execution environment is OpenJDK 1.8.0 and Ubuntu 18.04 with kernel version 5.4.0.

4.2 Benchmark

HiBench[15] is used in our experiments to collect performance metrics of the local Spark cluster under different configurations. HiBench is a big data benchmark suite that helps evaluate different big data frameworks such Hadoop, Spark and Flink. Specifically, after the execution of each application, HiBench will report the corresponding performance metrics such as execution time, throughput, etc. As listed in Table 1, we select four different Spark applications each with three different input datasets from HiBench in order to evaluate the effectiveness of DeepCAT under different workload characteristics.

4.3 Configuration Parameters

According to the official configuration guides as well as the empirical information extracted from our practical experiments, we finally select in total 32 configuration parameters that significantly influence performance. As shown in Table 2, there are 20 parameters from Spark, 7 parameters from YARN and 5 parameters from HDFS. For a Spark cluster, parameters of YARN are closely related to resource scheduling while HDFS parameters have a great influence on data read/write performance. Therefore, we should take these parameters into consideration in addition to Spark parameters to further improve the performance of a Spark application.

4.4 Baseline Algorithm

To evaluate the performance of DeepCAT when applied to online auto-tune configurations for big data frameworks, we compare it with two currently proposed online configuration auto-tuning approaches: a machine learning-based approach OtterTune and a DRL-based approach CDBTune. We omit the comparison with search-based approaches since they usually need a large number of time-consuming configuration evaluation and restart from scratch whenever a new tuning request comes.

OtterTune[26] is an online auto-tuning system that uses traditional machine learning models pipeline. It uses Gaussian process as its surrogate and Expected Improvement as its acquisition function. It recommends configuration through workload mapping to select similar workload data for modeling. We implement it through Python and the scikit-learn toolkit.

CDBTune[30] is the state-of-the-art work among DRL-based configuration auto-tuning systems, it uses DDPG as tuning agent to input internal metrics of the system and outputs proper configurations. We use the PyTorch library[23] to build neural networks to implement DDPG in CDBTune.

For fair comparison, we spend 3-4 days to generate enough samples and train the DRLs offline until they converged. Besides, we also feed thousands of offline samples to OtterTune to guarantee the quality of its machine learning model. In all experiments, we also set the number of total online tuning steps to be 5 according to CDBTune. After the online auto-tuning terminates, we record the best configurations ever found by DeepCAT, CDBTune and OtterTune and calculate the total time consumption during the 5 online tuning steps.

5 EXPERIMENTAL RESULTS AND ANALYSIS

5.1 Evaluation on Our Techniques

5.1.1 ***Effectiveness of RDPER.*** In order to fully utilizes the sparse but valuable close-to-optimal configurations in the high dimensional configuration space, we design a reward-driven prioritized experience replay mechanism for DeepCAT. To evaluate the effectiveness of RDPER, we first offline train TD3 separately with the conventional experience replay and RDPER on the same samples, and then execute 5 online tuning steps based on these two offline models. Figure 4 shows the execution time of TeraSort with D1 input dataset under the best configuration recommended online by each approach under different offline training iterations.

We can find that compared to TD3, the TD3 with RDPER approach is able to converge faster by a factor of 1.60 (2000 v.s. 3200) and recommend a much better configuration with 12.11% less execution time (37.0s v.s. 42.1s). More importantly, with only 800 iterations, it can find a configuration extremely close to the best configuration recommended by the TD3 approach with 3600 iterations. Therefore, through the RDPER mechanism, DeepCAT is able to take fully advantages of the sparse high-reward transitions and thus speeds up the offline training as well as improves configuration performance recommended during the online tuning phase.

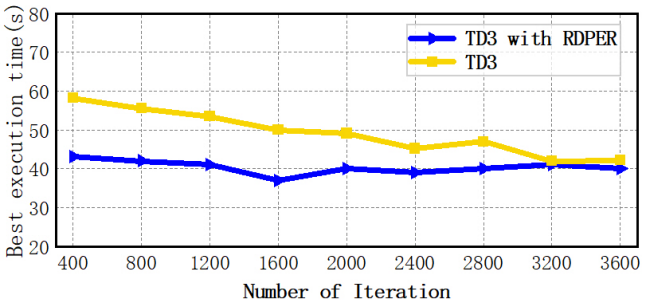

Figure 4: Execution time of the best configurations recommended by conventional TD3 and TD3 with RDPER under different offline training iterations.

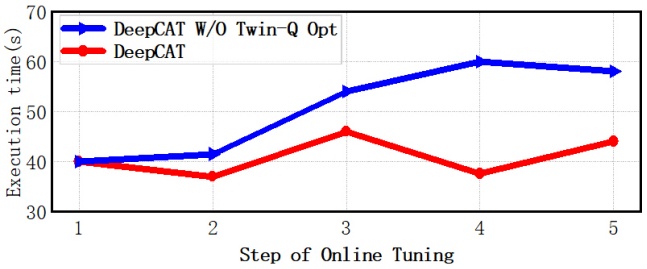

Figure 5: Execution time of configurations recommended by each online tuning step of DeepCAT and DeepCAT without the Twin-Q Optimizer.

5.1.2 ***Effectiveness of Twin-Q Optimizer.*** In order to reduce the total time consumption during the online tuning phase, Twin-Q Optimizer is utilized in DeepCAT to estimate the real execution time of a recommended configuration and optimize the sub-optimal ones. To evaluate its effectiveness, in this experiment, we run DeepCAT with and without the Twin-Q Optimizer to perform 5 sequential online tuning steps based on the same offline training model. Figure 5 shows the execution time of TeraSort with D1 under each configuration recommended during the online tuning phase.

As shown in the figure, with the Twin-Q Optimizer, DeepCAT is able to reduce the total time consumption of 5 online tuning steps by 19.29% (from 253.5 s to 204.6s) and find a better configuration with 7.29% less execution time. This significant improvement comes from the ability of Twin-Q Optimizer to estimate and optimize each online recommended action before real execution. For example, the original actions output by the latter four steps are all estimated as sub-optimal configurations in our experiments since they only gain modest Q-values from the twin critic networks. After that, our Twin-Q Optimizer modifies these original actions with a Gaussian noise until an estimated sub-optimal configuration is generated. As a result, the execution time under each step is reduced by 4.5s, 8.0s, 22.4s and 14.0s, respectively. Therefore, Twin-Q Optimizer is indeed able to help DeepCAT achieve a cost-efficient online configuration auto-tuning.

5.2 Comparison With Prior Approaches

5.2.1 ***Performance Speedup.*** In order to evaluate the effectiveness of DeepCAT for configuration auto-tuning, we compare the overall performance speedup over default configuration achieved by DeepCAT with CDBTune and OtterTune. As illustrated in Figure 6, these three approaches are all able to gain a great performance improvement for the 12 workload-input pairs over the default configuration. On average, DeepCAT, CDBTune and OtterTune is able to speed up the default performance by a factor of 4.66×, 3.21× and 2.82×, respectively. That is to say, with the same online tuning steps, the average speedups of DeepCAT over CDBTune and OtterTune are 1.45× and 1.65×. On the one hand, the GP regression model in OtterTune is too simple to capture the complex information necessary for the configuration auto-tuning problem, especially under the application scenarios without an obvious workload characteristics. On the other hand, based on DDPG, CDBTune works better than Ottertune in this high-dimensional configuration auto-tuning scenario. However, transitions with a high-performance configuration is quite rare and as a result, the prioritized experience replay mechanism based on TD error utilized in CDBTune still cannot achieve the efficient utilization of these sparse but valuable historical experiences. Different from CDBTune, DeepCAT trains TD3 with a reward-driven prioritized experience replay mechanism. It is able to address the value overestimation problem of DDPG and take fully advantages of the sparse but valuable transitions. Besides, the novel Twin-Q Optimizer can help DeepCAT estimate the real execution time of each recommended configuration during the online tuning phase and optimize the sub-optimal ones, which further improves the configuration performance.

It is worth noting that for the KMeans benchmark application, the average and maximized execution time speedup achieved by DeepCAT are respectively by a factor of 1.77× and 2.04× over CDBTune, and 1.98× and 2.17× over OtterTune. Specifically, as a machine learning workload, the KMeans application requires a large amount of memory to save the intermediate results of calculation and not enough memory may lead to OOM errors. As a result, high-reward transitions become more sparse, which magnifies the drawbacks of the DDGP-based approach CDBTune. In contrast, with the RDPER and Twin-Q Optimizer techniques, DeepCAT is still able to recommend an attractive configuration under the same constraint of total online tuning steps.

5.2.2 ***Online Tuning Cost.*** Considering the dynamic workload characteristics and hardware environment of a big data system, the time consumption consumed by the configuration auto-tuning approaches during the online tuning phase is also an vital indicator. Here we define the time consumption is the sum of the configuration evaluation time and the recommendation time. Figure 7 shows the total online tuning cost of each approach for the 12 workload-input pairs as well as the time breakdown. Specifically, the total time consumed during the online tuning phase by DeepCAT is reduced by 24.64% on average and up to 50.08% compared with CDBTune, while 39.71% on average and up to 53.39% compared with OtterTune. This non-trivial improvement is because that different from CDBTune and OtterTune, the Twin-Q Optimizer in DeepCAT is able to estimate the execution time of each recommended action with no need for the costly real execution and optimize the original sub-optimal actions. Considering the performance speedup achieved by DeepCAT as described in the previous subsection, DeepCAT is indeed able to achieve a cost-efficient online configuration auto-tuning for big data frameworks.

Next, we will give a short discussion about the time consumed on configuration recommendation of each approach during the 5 online steps. The recommendation time of DeepCAT and CDBTune is respectively 0.69s and 0.25s, which is almost negligible for the tens-of-minutes long total online tuning time. For DRL-based approaches, obtain the output of the actor and critic networks to recommend an action is relatively cheap, even considering the Twin-Q Optimizer utilized in DeepCAT. On the contrary, once a configuration tuning request is arrived, OtterTune needs to perform a time-consuming GP model training first according to the workload characteristics and then starts the configuration recommendation. The total recommendation time during the whole online tuning phase thus reach a noticeable 43.25 seconds.

5.2.3 ***Effectiveness under Tuning Cost Constraint.*** For real online configuration auto-tuning applications, there is usually a

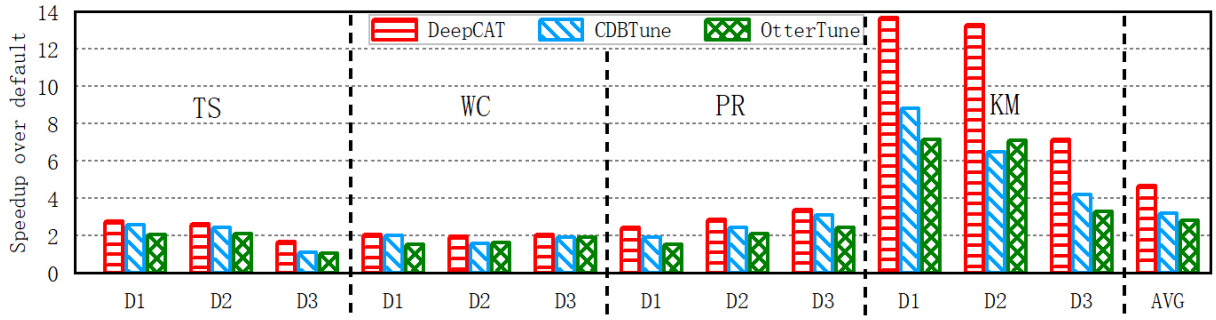

Figure 6: Speedup of best configurations recommended by different approaches over the default configuration (higher is better). Input datasets (D1, D2, D3) for each workload is corresponding with the description in Table 1.

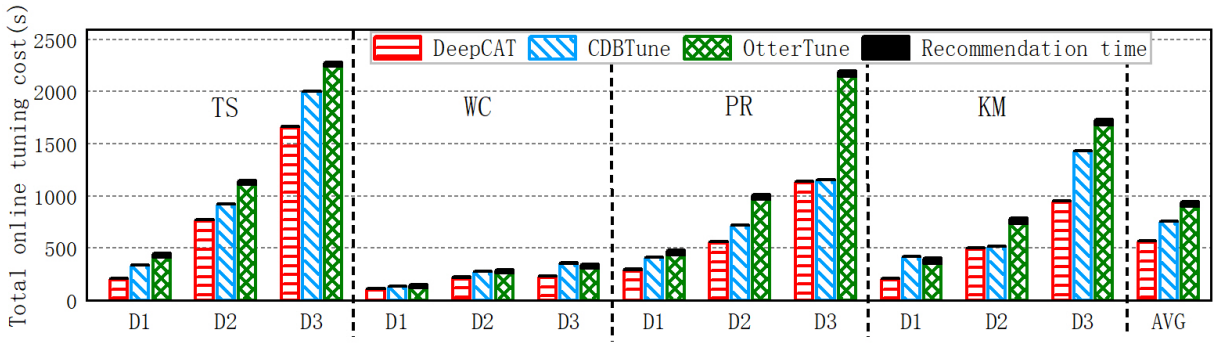

Figure 7: Comparison of the total online tuning time consumed by different approaches (lower is better). We mark the recommendation time of each approach black.

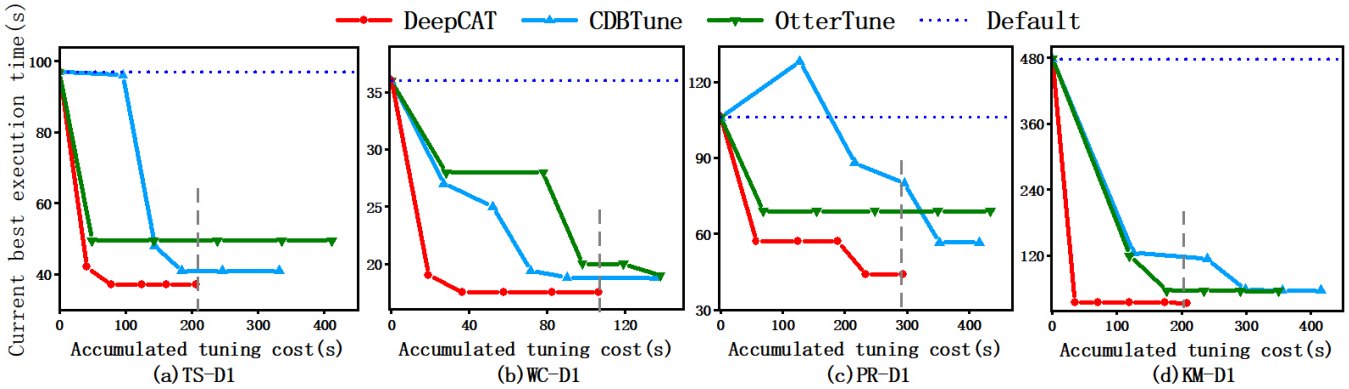

Figure 8: The execution time of current best configuration and the corresponding accumulated tuning cost along with the 5 online tuning steps of DeepCAT, CDBTune, OtterTune.

user-specified constraint on the total online tuning time consumption. Therefore, in this subsection, we discuss the the effectiveness of DeepCAT under an online tuning cost constraint. Figure 8 illustrates the accumulated tuning cost along with the 5 online tuning steps and the corresponding performance of current best configuration found by DeepCAT, CDBTune and OtterTune. Significantly, compared with CDBTune and OtterTune, DeepCAT is able to find a better configuration with much less accumulated tuning time for all the workloads. In fact, with the Twin-Q Optimizer, most of the recommended actions by DeepCAT have an excellent execution time and thus DeepCAT can orchestrate a stable online tuning phase. As a result, it is reasonable to believe that under the same constraint of total online tuning time, DeepCAT is able to perform more tuning steps and thus recommend a better configuration.

5.3 Evaluation on Adaptability

5.3.1 ***Varying Different Workloads.*** In practice, the workload characteristics of a big data system is usually time-varying and it is infeasible to offline train different model for each workload. Therefore, in this experiment, we evaluate the the ability of DeepCAT to adapt to different workloads. To this end, we use DeepCAT, OtterTune and CDBTune to respectively perform online configuration auto-tuing for PageRank based on the models offline trained under different workloads. Figure 9 shows the execution time of PageRank with D1 input dataset under the best configuration recommended online and the total online tuning cost. It is worth noting that we utilize $M\_TS \rightarrow PR$ to indicate applying the model trained on TS workload to online tuning PR.

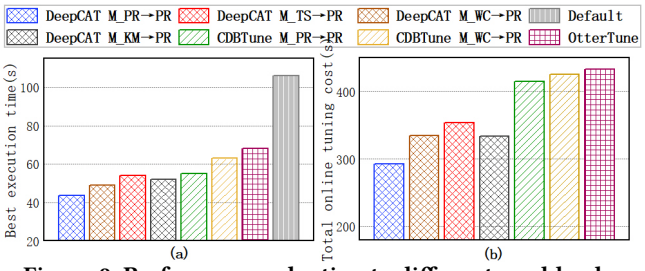

Figure 9: Performance adapting to different workloads.

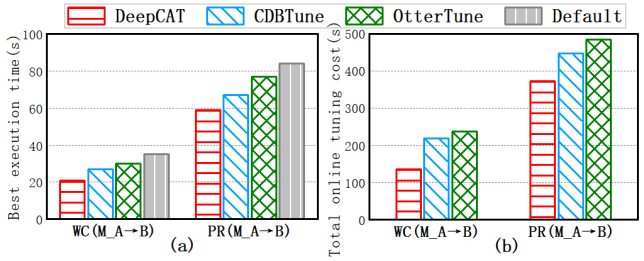

Figure 10: Performance adapting to different hardware environments.

First of all, DeepCAT trained on other workloads can still recommend a satisfying configuration with only 11.22% to 19.44% more execution time compared with the best configuration found by DeepCAT directly trained on PageRank (the blue bar). Besides, it is worth noting that DeepCAT trained on other workloads all outperform CDBTune and OtterTune, even they are specifically trained for PageRank. In detail, these four DeepCAT models achieve an average 15.86% and 27.21% performance improvement than CDBTune and OtterTune, while reducing the total online tuning cost by 21.67% and 24.02% on average and up to 30.95% and 36.03%, respectively. The reason is that DeepCAT is able to learn a better policy network through TD3 and the RDPER mechanism for the original environment. When this offline model is utilized during the online tuning stage for another workload, the good adaptability of DRL as well as the Twin-Q Optimizer in DeepCAT together guarantee the performance of recommended configurations. Experimental results indicate that DeepCAT indeed has a good adaptability when the workload changes, which is common for big data frameworks.

Second, $M\_TS \rightarrow PR$ (the red bar) performs worst among all the DeepCAT models. In fact, the TeraSort application is a CPU and memory-intensive workload based on map and reduce processes, which presents the most different workload characteristics from PageRank based on iterative selections. Even so, we can still find that DeepCAT successfully recommend a better configuration with much fewer total online tuning time compared with CDBTune and OtterTune.

5.3.2 ***Varying Different Hardware Environments.*** In addition to workload characteristics, the hardware environment of a big data system is also time-varying. In order to evaluate the adaptability of DeepCAT to different hardware environment, we created a VM-based Spark cluster which also contains three nodes, with a total of 24 CPU cores, 24G memory and 150G disk. We denote this new environment as Cluster-B and the former environment described in Section 4.1 as Cluster-A. After that, DeepCAT, CDBTune and

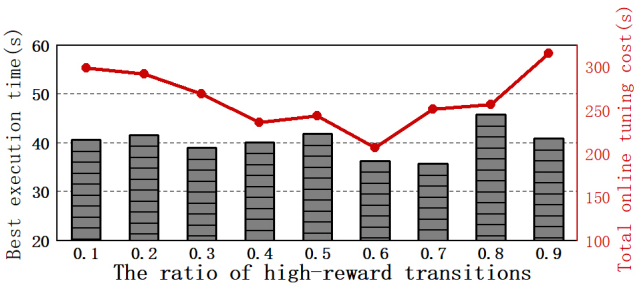

Figure 11: The performance of DeepCAT under different β settings.

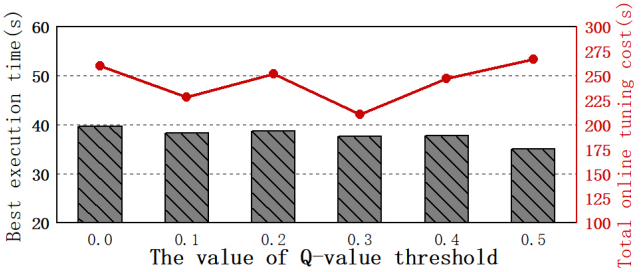

Figure 12: The performance of DeepCAT under different Q_{th} settings.

OtterTune are trained on cluster-A and then perform online configuration auto-tuning for WordCount and PageRank with D1 input datasets on cluster-B. Note that if the recommended configuration parameters are outside the scope of the new environment, we need to clip it to the boundary to avoid unnecessary waste of resources. As shown in Figure 10, DeepCAT performs best among all these online configuration auto-tuning methods. For the WordCount application, the speedup of execution time over default achieved by DeepCAT, CDBTune and OtterTune are respectively by a factor of 1.68×, 1.30× and 1.17×. While for the PageRank application, the speedup ratios are 1.42×,1.25× and 1.09×. In the meanwhile, DeepCAT consumes the less total online tuning cost. These results verify that DeepCAT also has a strong adaptability to the hardware environment.

5.4 Hyper-Parameters in DeepCAT

5.4.1 ***High-reward transition ratio (β).*** The quality of offline training has a great impact on both the effectiveness and efficiency of an online configuration auto-tuning approach. In order to improve the learned policy network in the sparse positive reward scenario, DeepCAT utilizes RDPER mechanism in the offline training stage. Since β controls the ratio of high-reward transitions in the total replayed batch samples for TD3 training, here we discuss how to determine it in our experiments. We set the value of β from 0.1 to 0.9 and increase by 0.1 each time and finally we train 9 offline models for online tuning TeraSort with D1 dataset. Figure 11 shows the relationship between β and best execution time achieved by DeepCAT during the online tuning stage. We can find that no matter the ratio β is too high or too low, DeepCAT will fail to recommend a close-to-optimal configuration in a short time. The reason is that training network with a batch of all-good or all-bad data may both lead to model over-fitting. On the other hand, DeepCAT

performs relatively better when the value of β is between 0.4 to 0.7. Considering both the best configuration and the total online total cost, we finally set β to be 0.6 in our experiments.

5.4.2 ***Q-value threshold (Q_{th}).*** The hyper-parameter is used to identify sub-optimal actions in the Twin-Q Optimizer, which is vital to reduce the total online tuning cost. A larger Q_{th} will increase the exploration around the sub-optimal space, while a smaller Q_{th} tends to exploit the good configuration has been already found. In this experiment, the value range of Q_{th} is set according to the approximate reward of the target configuration performance. As shown in Figure 12, although $Q_{th} = 0.5$ is able to recommend the best configuration, its total online tuning cost is the largest in the meanwhile. This is because a larger Q-value threshold will drive more risky explorations during the online tuning stage, which is a double-edged sword. Therefore, we set $Q_{th} = 0.3$ in our experiments because it consumes the least total time and can recommend a close-to-optimal configuration with only a slight 2.54 seconds more execution time compared with $Q_{th} = 0.5$.

6 RELATED WORK

In the past few years, there are already some literatures focusing on configuration auto-tuning for different software systems. Broadly speaking, prior studies can be mainly divided into three categories: search-based, machine learning-based and reinforcement learning-based.

(1) Search-based methods explore the configuration space according to specific rules and run on the same workload with different configuration settings until they find satisfying results. BestConfig [32] utilizes a divide-and-diverge sampling method as well as a recursive bound-and-search algorithm to automatically tune system configurations within a resource limit. However, these random-based methods are not able to make full use of the valuable information from historical observation sets. Due to the effectiveness in solving black-box optimization problem, Bayesian Optimization (BO) is widely utilized to address the configuration auto-tuning problem[1, 7, 8, 16, 17]. Although they are simple to run and do not require much knowledge of the system, they still require a high number of the time-consuming configuration evaluation and thus are not suitable for online configuration auto-tuning. To reduce the tuning cost, Lynceus[6] uses a long-sighted and budget-aware technique to determine which configurations to test by predicting the long-term impact of each exploration search space and early stops the execution of a job on sub-optimal configurations. However, the model is not accurate in the early stage due to the lack of historical data, which will lead to misjudging the best configuration. Hence, it may not be effective until dozens of iterations. Furthermore, ResTune[31] leverages the tuning experience from the history tasks and transfers the accumulated knowledge to accelerate the tuning process of the new tasks. But it does not consider the evaluation cost of sub-optimal configurations. With the Twin-Q Optimizer, DeepCAT is able to identify sub-optimal configurations and optimize them to reduce the total online tuning cost.

(2) Learning-based methods utilize the machine learning techniques to tune configurations. For example, OtterTune[26] with Lasso-based knob selection and map unseen database workloads to previous workloads and recommend configuration by a traditional machine learning model. Other methods [3, 4, 11, 29] also rely on machine learning algorithms to train a performance prediction model and then find the optimal configuration using a search method such as genetic algorithm. Although these studies is able to employ performance models to avoid real configuration evaluation during online tuning phase, they seriously rely on large-scale high-quality training samples and have a poor adaptability to environment changes. In contrast, DeepCAT can fully utilize the sparse but valuable transitions and have a good adaptability to the time-varing workload and hardware.

(3) Reinforcement learning-based methods treat configurations tuning as a trial-and-error process based on a reward signal and learn from the interactions with the environment. Hence many RL algorithms have been successfully utilized in system configuration auto-tuning[5, 12, 22]. CDBTune[30] is the first to utilize a DRL model to recommend configurations. Insprided by CDBTune, Qtune[18] uses a double-state DDPG and can provide better fine-grained tuning. Specifically, it uses a neural network to predict the change value ΔS of the state after processing a query in the database. However, if the prediction of ΔS is slightly biased, the cumulative error will be thousands of miles away. On the other hand, WATuning[10] designs an attention-based DDPG that consider the impact of different workloads on the state of the system. It also designs a multi-instance mechanism that classifies workloads in a fine-grained manner. However, it requires a large amount of training datasets and it may not work well on tuning complex and diverse workloads in big data systems. In contrast, DeepCAT utilizes the TD3 algorithm instead of DDPG to alleviate value overestimation. More importantly, DeepCAT utilizes a novel reward-driven prioritized experience replay mechanism to fully utilize the rare but valuable experiences and a Twin-Q Optimizer algorithm to achieve a low-cost exploration-exploitation trade off.

7 CONCLUSION AND FUTURE WORK

In this paper, we propose a cost-efficient online configuration auto-tuning approach named DeepCAT for big data frameworks. To reduce the total online tuning cost, DeepCAT: 1) utilizes the TD3 algorithm instead of DDPG to alleviate value overestimation; 2) designs a novel reward-driven PER mechanism to fully utilize the rare but valuable experiences; 3) proposes a Twin-Q Optimizer to estimate the execution time of each action without the costly configuration evaluation and optimize the sub-optimal ones to achieve a low-cost exploration-exploitation trade off. Experimental results based on a local Spark cluster and HiBench benchmark applications show the effectiveness and efficiency of DeepCAT compared with CDBTune and OtterTune.

Recently, LOCAT[28] and LITE[20] are proposed to reduce the tuning cost through white-box analysis of the target application. We also acknowledge the function of software analysis for cost-efficient configuration tuning and these inspiring works are able to further enhance our approach. How to utilize software analysis methods to further reduce the online tuning cost is one of our future work.

ACKNOWLEDGMENTS

The authors would like to thank the anonymous reviewers for their valuable comments and suggestions. This work was supported by the Key-Area Research and Development Program of Guangdong Province under Grant 2020B010165002, and the National Natural Science Foundation of China under Grant 61902440, 61872002 and U1811462.

REFERENCES

[1] Omid Alipourfard, Hongqiang Harry Liu, Jianshu Chen, Shivaram Venkataraman, Minlan Yu, and Ming Zhang. 2017. {CherryPick}: Adaptively Unearthing the Best Cloud Configurations for Big Data Analytics. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. 469–482.

[2] Liang Bao, Xin Liu, Fangzheng Wang, and Baoyin Fang. 2019. ACTGAN: automatic configuration tuning for software systems with generative adversarial networks. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 465–476.

[3] Liang Bao, Xin Liu, Ziheng Xu, and Baoyin Fang. 2018. Autoconfig: Automatic configuration tuning for distributed message systems. In *2018 33rd IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 29–40.

[4] Zhendong Bei, Zhibin Yu, Ni Luo, Chuntao Jiang, Chengzhong Xu, and Shengzhong Feng. 2018. Configuring in-memory cluster computing using random forest. *Future Generation Computer Systems* 79 (2018), 1–15.

[5] Xiangping Bu, Jia Rao, and Cheng-Zhong Xu. 2009. A reinforcement learning approach to online web systems auto-configuration. In *2009 29th IEEE International Conference on Distributed Computing Systems*. IEEE, 2–11.

[6] Maria Casimiro, Diego Didona, Paolo Romano, Luis Rodrigues, Willy Zwaenepoel, and David Garlan. 2020. Lynceus: Cost-efficient tuning and provisioning of data analytic jobs. In *2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 56–66.

[7] Hui Dou, Pengfei Chen, and Zibin Zheng. 2020. Hdconfigor: automatically tuning high dimensional configuration parameters for log search engines. *IEEE Access* 8 (2020), 80638–80653.

[8] Ayat Fekry, Lucian Carata, Thomas Pasquier, Andrew Rice, and Andy Hopper. 2020. To tune or not to tune? in search of optimal configurations for data analytics. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2494–2504.

[9] Scott Fujimoto, Herke Hoof, and David Meger. 2018. Addressing function approximation error in actor-critic methods. In *International conference on machine learning*. PMLR, 1587–1596.

[10] Jia-Ke Ge, Yan-Feng Chai, and Yun-Peng Chai. 2021. WATuning: A Workload-Aware Tuning System with Attention-Based Deep Reinforcement Learning. *Journal of Computer Science and Technology* 36, 4 (2021), 741–761.

[11] Yijin Guo, Huasong Shan, Shixin Huang, Kai Hwang, Jianping Fan, and Zhibin Yu. 2021. GML: Efficiently Auto-Tuning Flink's Configurations Via Guided Machine Learning. *IEEE Transactions on Parallel and Distributed Systems* 32, 12 (2021), 2921–2935.

[12] Xue Han and Tingting Yu. 2020. Automated performance tuning for highly-configurable software systems. *arXiv preprint arXiv:2010.01397* (2020).

[13] Haochen He, Zhouyang Jia, Shanshan Li, Yue Yu, Chenglong Zhou, Qing Liao, Ji Wang, and Xiangke Liao. 2021. Multi-Intention Aware Configuration Selection for Performance Tuning. (2021).

[14] Yigong Hu, Gongqi Huang, and Peng Huang. 2020. Automated reasoning and detection of specious configuration in large systems with symbolic execution. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. 719–734.

[15] Shengsheng Huang, Jie Huang, Jinquan Dai, Tao Xie, and Bo Huang. 2010. The HiBench benchmark suite: Characterization of the MapReduce-based data analysis. In *2010 IEEE 26th International Conference on Data Engineering Workshops (ICDEW 2010)*. IEEE, 41–51.

[16] Pooyan Jamshidi and Giuliano Casale. 2016. An Uncertainty-Aware Approach to Optimal Configuration of Stream Processing Systems. In *2016 IEEE 24th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*. 39–48. https://doi.org/10.1109/MASCOTS.2016.17

[17] Md Muhib Khan and Weikuan Yu. 2021. ROBOTune: High-Dimensional Configuration Tuning for Cluster-Based Data Analytics. In *50th International Conference on Parallel Processing*. 1–10.

[18] Guoliang Li, Xuanhe Zhou, Shifu Li, and Bo Gao. 2019. Qtune: A query-aware database tuning system with deep reinforcement learning. *Proceedings of the VLDB Endowment* 12, 12 (2019), 2118–2130.

[19] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. 2015. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971* (2015).

[20] Chen Lin, Junqing Zhuang, Jiadong Feng, Hui Li, Xuanhe Zhou, and Guoliang Li. 2022. Adaptive Code Learning for Spark Configuration Tuning. ICDE.

[21] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602* (2013).

[22] Ting-Yu Mu, Ala Al-Fuqaha, and Khaled Salah. 2019. Automating the configuration of MapReduce: A reinforcement learning scheme. *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 50, 11 (2019), 4183–4196.

[23] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems* 32 (2019).

[24] David Buchaca Prats, Felipe Albuquerque Portella, Carlos HA Costa, and Josep Lluis Berral. 2020. You Only Run Once: Spark Auto-Tuning From a Single Run. *IEEE Transactions on Network and Service Management* 17, 4 (2020), 2039–2051.

[25] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. 2015. Prioritized experience replay. *arXiv preprint arXiv:1511.05952* (2015).

[26] Dana Van Aken, Andrew Pavlo, Geoffrey J Gordon, and Bohan Zhang. 2017. Automatic database management system tuning through large-scale machine learning. In *Proceedings of the 2017 ACM international conference on management of data*. 1009–1024.

[27] Christopher JCH Watkins and Peter Dayan. 1992. Q-learning. *Machine learning* 8, 3 (1992), 279–292.

[28] Jinhan Xin, Kai Hwang, and Zhibin Yu. 2022. LOCAT: Low-Overhead Online Configuration Auto-Tuning of Spark SQL Applications [Extended Version]. *arXiv preprint arXiv:2203.14889* (2022).

[29] Zhibin Yu, Zhendong Bei, and Xuehai Qian. 2018. Datasize-aware high dimensional configurations auto-tuning of in-memory cluster computing. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*. 564–577.

[30] Ji Zhang, Yu Liu, Ke Zhou, Guoliang Li, Zhili Xiao, Bin Cheng, Jiashu Xing, Yangtao Wang, Tianheng Cheng, Li Liu, et al. 2019. An end-to-end automatic cloud database tuning system using deep reinforcement learning. In *Proceedings of the 2019 International Conference on Management of Data*. 415–432.

[31] Xinyi Zhang, Hong Wu, Zhuo Chang, Shuowei Jin, Jian Tan, Feifei Li, Tieying Zhang, and Bin Cui. 2021. Restune: Resource oriented tuning boosted by meta-learning for cloud databases. In *Proceedings of the 2021 International Conference on Management of Data*. 2102–2114.

[32] Yuqing Zhu, Jianxun Liu, Mengying Guo, Yungang Bao, Wenlong Ma, Zhuoyue Liu, Kunpeng Song, and Yingchun Yang. 2017. Bestconfig: tapping the performance potential of systems via automatic configuration tuning. In *Proceedings of the 2017 Symposium on Cloud Computing*. 338–350.