CTuner: Automatic NoSQL Database Tuning with Causal Reinforcement Learning

Genting Mai
Zilong He
Guangba Yu
Sun Yat-sen University
Guangzhou, China
maigt3@mail2.sysu.edu.cn
hezlong@mail2.sysu.edu.cn
yugb5@mail2.sysu.edu.cn

Zhiming Chen
Pengfei Chen*
chenzhm37@mail2.sysu.edu.cn
chenpf7@mail.sysu.edu.cn

ABSTRACT

The rapid development of information technology has necessitated the management of large volumes of data in modern society, leading to the emergence of NoSQL databases (e.g., MongoDB). To meet the huge demand for efficient data management and query, optimizing the performance of these databases has become crucial. Currently, some reinforcement learning-based methods have been used to improve the efficiency of databases by tuning customizable database configurations. However, these methods have limitations: they ignore operating system configurations, incur high training costs with more knobs, and adapt poorly to new environments with varying workloads and hardware. To address these issues, we propose a novel and effective approach named CTuner for the online performance tuning of NoSQL databases. CTuner skips cold start by Bayesian optimization-based learning, and improves the exploitation strategy of the Twin Delayed Deep Deterministic Policy Gradient (TD3) model with causal inference. Practical implementation and experimental evaluations on three prominent NoSQL databases show that CTuner can find a better configuration at the same time cost than state-of-the-art approaches, with up to a 27.4% improvement in throughput and up to 13.2% reduction in 95%-tail latency. Moreover, we introduce meta-learning to enhance the adaptability of CTuner and confirm that it is able to reliably improve performance under new environments and workloads.

CCS CONCEPTS

• **Software and its engineering → Software performance**.

KEYWORDS

Configuration tuning, NoSQL database, Reinforcement learning, Meta-learning, Causal inference.

ACM Reference Format:
Genting Mai, Zilong He, Guangba Yu, Zhiming Chen, and Pengfei Chen. 2024. CTuner: Automatic NoSQL Database Tuning with Causal Reinforcement Learning. In *15th Asia-Pacific Symposium on Internetware (Internetware 2024), July 24–26, 2024, Macau, China.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3671016.3674809

Internetware 2024, July 24–26, 2024, Macau, China

ACM ISBN 979-8-4007-0705-6/24/07
https://doi.org/10.1145/3671016.3674809

1 INTRODUCTION

The proliferation of large-scale data in modern society has been a driving force behind the rapid advancement of databases, particularly NoSQL databases. NoSQL databases have found extensive applicability in various domains, including web applications, big data analysis, and Internet of Things (IoT) platforms [40]. With the ever-increasing demand for effective data management and analysis, optimizing database performance is crucial to enhance data retrieval, storage, and processing capabilities, ultimately enabling organizations and individuals to gain a competitive edge in the data-driven era.

Database performance relies on a range of knobs (i.e., configuration parameters or options) that exhibit both explicit and implicit dependencies, making manual configurations based on expert experience of database administrators (DBAs) labor-intensive and error-prone [50] , which further exacerbates the complexity of the problem.

Previous attempts at automatic database configuration tuning can be divided into two major classes, namely Bayesian Optimization (BO) based [14, 15, 24, 45, 52, 53] and Reinforcement Learning (RL) based [7, 18, 20, 29, 48, 50, 54] methods. Given an objective function that models knobs and performance metrics (e.g., throughput), the basic idea of BO-based methods is to find the optimal configurations (i.e., set of knobs' values) with the best performance based on a probabilistic model. However, BO-based methods struggle to rapidly search for optimal configurations due to the exponential growth of the configuration space [14], limited adaptation to new workloads and environments [32] and the need for continuous model updates and tuning efforts [11].

RL-based methods effectively address NoSQL database tuning by treating it as a sequential decision-making problem and utilizing policy-based model-free RL agents. These methods leverage deep neural networks to capture complex relationships between knobs and performance metrics, making them suitable for high-dimensional configuration spaces [18, 20, 29, 50]. Therefore, existent RL-based approaches have shown superior effectiveness compared to manual [51] and BO-based tuning methods [13]. However, they still possess significant limitations.

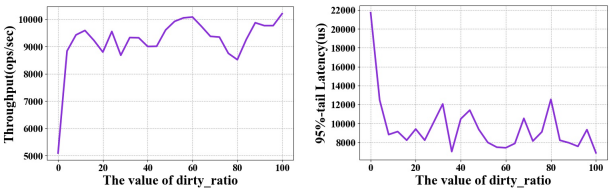


Figure 1: The performance of MongoDB in *workload_g* that only changes the value of the OS knob *dirty_ratio*.

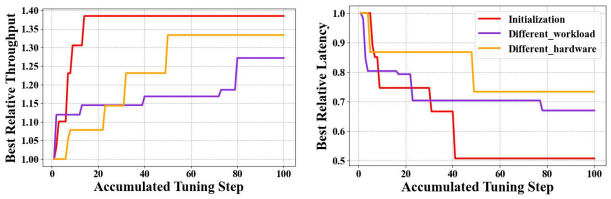


Figure 2: Comparison of the best relative throughput and latency of DDPG under different workload or hardware for MongoDB, where *Initialization* represents online tuning in *workload_g*, *Different_workload* is online tuning in *workload_f*, *Different_hardware* is online tuning in *workload_g* and another hardware with different CPU configuration.

Limitation1: Ignoring Operating System Configurations. The performance of a database can be influenced by OS configurations. We conducted a performance experiment on MongoDB under the *workload_g* from YCSB [10]. Changing only the OS knob *dirty_ratio* from 0 to 100 while keeping other knobs at default values, we observed significant changes. MongoDB's throughput increased from 5096 ops/s to 10204 ops/s, and the 95%-tail latency decreased from 21715 us to 6891 us (Fig. 1). This knob, a Linux kernel parameter, limits dirty pages in memory, directly impacting MongoDB's performance. Therefore, considering the OS configuration expands the tuning space and improves database performance. However, existing RL-based methods for NoSQL database performance tuning generally overlook this aspect [21].

Limitation2: Training cost increases significantly with the number of considered knobs. Modern NoSQL databases offer numerous tunable knobs. For example, Apache Cassandra has 155 configurable knobs [50]. To ensure the effectiveness of RL-based methods, it is intuitive and necessary to consider as many knobs as possible, leading to a large sample size and significant testing overhead [44]. However, the vast configuration space makes it challenging for RL-based methods to converge, resulting in increased training costs [27]. Therefore, reducing the configuration space via pre-selecting the set of important knobs for tuning is still an urging problem.

Limitation3: Weak adaptability for new environments with different workloads and hardware. Deploying one type of database in varied environments is a common practice. However, existing RL-based approaches lack adaptability in diverse environments. We use DDPG for offline training in *workload_g*. Then, we perform online tuning in *workload_g* and *workload_f*, and tune in a different hardware environment (Fig. 2). Results show poor performance in unseen environments and high time consumption for fine-tuning. Therefore, improving model generalization and reducing tuning time is crucial.

Our approach. To address the limitations of existing methods, we propose CTuner [1], a reinforcement learning-based **Hybrid Tuning System** for NoSQL databases, which rapidly adapts to environments with unseen hardware and workloads. It considers both OS and NoSQL database knobs (**Limitation1 solved**). CTuner consists of offline training and online tuning. Offline training is a two-stage task. In the first stage, The Hash-enhanced Subspace BO (HeSBO) [33] is used to generate high-quality training samples. Bayesian optimization can quickly find better configuration with fewer iterations. But its learning potential is limited. Therefore, the second stage is introduced, we use Twin Delayed Deep Deterministic Policy Gradient (TD3) [17] as the Deep Reinforcement Learning (DRL) model for tuning. The samples from the previous stage are used to solve the cold start problem of DRL. At the same time, Random Forest [6] is used to reduce the input dimension of the neural network to lower the training costs. Besides, CTuner innovatively integrates causal inference and meta-learning [43] into DRL. Causal inference can learn the causal relationship between knobs and performance to speed up the convergence of the model (**Limitation2 solved**). And meta-learning improves the model's ability to adapt when encountering unseen environments (**Limitation3 solved**).

In addition, online tuning consists of three parts: Collector, Adaptor, and Rule Filter. Collector gathers NoSQL databases environment information. Adaptor can quickly adapt the model to unseen environments and generate better configurations. Rule Filter adjusts knob values based on user-defined rules. This combination enables a secure and optimal configuration quickly. Compared with the state-of-the-art methods, our experimental results show that CTuner performs better given the same time and resource cost.

Contributions. We summarize our contributions as follows:

- We propose an online tuning framework for NoSQL databases with safer configurations, low tuning cost and high efficiency, considering both NoSQL databases and OS configurations.
- We combine Bayesian optimization, causal inference, and meta-learning to extend traditional reinforcement learning methods to improve the efficiency and adaptability of the online learning model.
- Practical implementation and experimental evaluations on three prominent NoSQL databases show that CTuner can find a better configuration at the same time cost than state-of-the-art approaches, with up to a 27.4% improvement in throughput and up to 13.2% reduction in 95%-tail latency.

2 OVERVIEW OF CTUNER

The goal of CTuner is to reduce the time of online tuning while achieving optimal NoSQL databases performance. CTuner provides a novel architecture, which is shown in Fig. 3. CTuner consists of (a) **offline part**, which trains a tuning model that can quickly adapt to different environments, and (b) **online part**, which recommends good configuration for unseen environments according to user requirements.

2.1 Offline Training

Booster. In the first stage, CTuner uses HesBO in multiple training environments to collect high-quality training samples. Then, these samples are trained through Random Forest, resulting in a ranked list of knobs by importance. We take the top-ranked knobs in the list

[1]The source code is available at https://github.com/IntelligentDDS/Ctuner.git

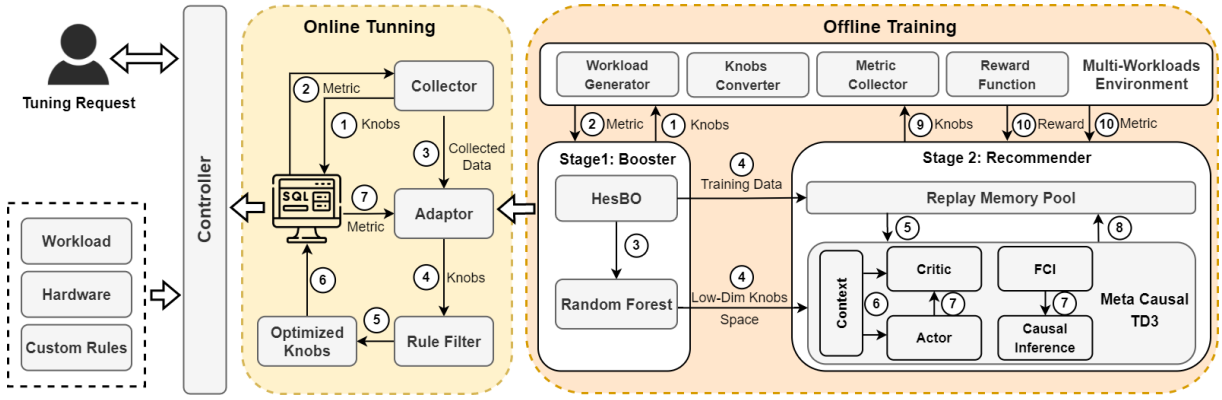


Figure 3: The architecture of CTuner

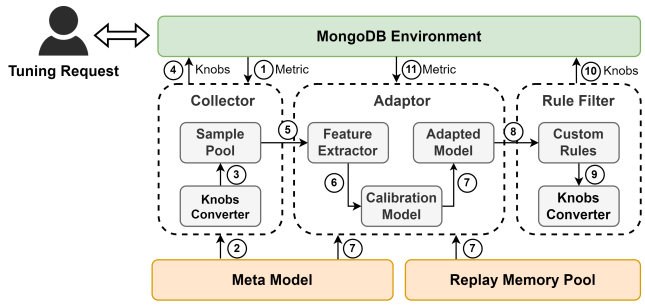


Figure 4: Online tuning of MongoDB using CTuner.

as the training configuration space, reducing the control dimension of the model.

Recommender. In the second stage, CTuner designs a TD3 model based on meta-learning and causal inference, which uses dynamic workload to train multiple meta-training tasks. A multi-task priority experience pool is designed for storing and extracting historical data. Unlike traditional fine-tuning methods, meta-learning allows models to quickly adapt to new workloads. Next, CTuner warms up the model with pre-generated high-quality samples to accelerate convergence. During training, CTuner uses GRU [9] to extract meta-features of workload and hardware, learning feature knowledge from different environments. Besides using actors for configuration exploration and exploitation, Fast Causal Inference (FCI) [42] is applied to historical training samples to generate a causal graph. Then causal inference selects the knob with the largest causal effect for tuning, maximizing the use of high-quality historical data.

2.2 Online Tuning

Collector. In the early stage of tuning, CTuner uses the model trained offline to generate configurations. And it is deployed to get metric samples including state and performance. Each sample is shaped like $\{a, s, r\}$, where a is a set of knobs as well as their values, s is the database state, r is the performance including throughput and latency. After that, these samples are fed into the Adaptor.

Adaptor. After receiving the sample S from the Collector, the Adapter extracts the meta features of the environment. These are trained together with data from the Reply Memory Pool to obtain a covariate offset-corrected model. It adjusts the offline model to the current environment. This module can save tuning time and allow the tuning model to find better configurations faster.

Rule Filter. This module allows users to customize the range of configuration parameters based on expert experience. This makes the recommended configuration more safe and reliable. Moreover, expert users use this function to improve the recommendation efficiency of the model.

2.3 An Illustrative Example

Fig. 4 illustrates the process of online tuning of MongoDB using CTuner. Users can input the range of configuration parameters into the system based on expert experience. CTuner uses the meta-model trained offline to generate configuration to obtain the corresponding metric and store it in the sample pool. After several times of collections, these data are sent from Collector to Adaptor. The Adaptor extracts features from the user's hardware and workload through the feature extractor. Then, these features are used to fit the historical data in the replay memory pool to get a calibration model, and Adapted Model is obtained by adjusting the offline trained model. Configuration is generated using the model and sent to the rule filter. Finally the selected configuration will be applied into the MongoDB cluster. If the obtained performance satisfies the user's requirements or reaches the maximum number of tuning iterations, the configurations are returned to the user. Otherwise, the metric is fed back to the Adaptor to generate configuration continuously.

3 DESIGN OF CTUNER

CTuner consists of several critical components, each of which plays an important role in realizing a tuning system with very few samples, low cost and high adaptability. (a) CTuner considers both operating system and NoSQL database knobs (**Limitation1 solved**). (b) CTuner uses BO to sample high-quality tuning data, and use random forest to filter out important knobs to reduce the configuration space. CTuner designs causal inference algorithms to make full use of high-quality samples to speed up model convergence (**Limitation2 solved**). (c) CTuner designs meta reinforcement learning algorithms to improve model adaptability (**Limitation3 solved**). We describe the components of CTuner in detail.

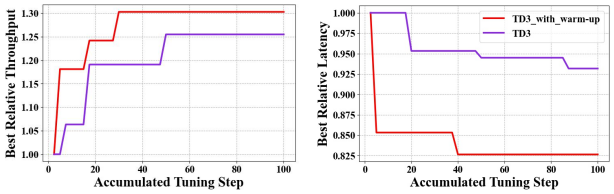


Figure 5: Comparison of the best relative throughput and latency of TD3 with and without warm-up for Redis

3.1 BO for High-quality Sampling

Bayesian optimization (BO) is mainly used to solve computationally expensive black-box optimization problems. Specifically, at each iteration, the prior knowledge observed before is used for the next optimization. The two key components of BO include the surrogate model and acquisition function. BO is very suitable for solving NoSQL configuration optimization problems where the objective function is unknown and the calculation is complex.

The traditional BO algorithm is easy to fall into the local optimal solution [14]. Although extended BO is able to jump out of local optimal solutions, its exploration capability may be limited compared with RL [47]. Therefore, we choose to combine the advantages of RL and BO to achieve better optimization results. Moreover, the cold start problem is very prominent for RL. It can be seen from Fig. 5 that the tuning effect of TD3 with warm-up and without warm-up is very different. High-quality samples at the early stage of training can solve the cold start problem and speed up training. For NoSQL database tuning tasks, obtaining a sample requires a long period of configuration deployment and pressure testing. It leads to prohibitively high training costs. If the obtained samples are of high quality, fewer samples can also allow the neural network to learn enough knowledge. It can be seen from Fig. 6 that compared with the random algorithm and LHS [22], BO can learn the features in NoSQL databases more quickly and obtain higher quality samples. Therefore, we choose BO to generate high-quality samples, and then warm-up the reinforcement learning model to make it converge faster and get better tuning results [7].

Considering that traditional BO cannot accept high-dimensional input space. Inspired by LlamaTune [26], we use Hash-enhanced Subspace BO (HeSBO) [33], which is a random linear projection variant. Assuming that the original D-dimensional search space is $\mathbb{H}$ and the target dimension is d. HeSBO defines the d-dimensional search space $\mathbb{L} = [-1, 1]^d$. Next, it generates a random projection matrix $M \in R^{D \times d}$. Each row of M contains a non-zero element in a column that is set to ±1. Both values and columns are chosen randomly. Essentially, M provides a one-to-many mapping. Each original parameter in $\mathbb{H}$ is controlled by a synthesized parameter in $\mathbb{L}$, and each synthesized parameter can control multiple original parameters.

After the dimension transformation, it can be processed with ordinary BO. Like most BO methods [8, 12], we also choose Gaussian Process (GP) as the surrogate model. The Expected Improvement (EI) acquisition function is used to recommend the next better configuration. NoSQL database cluster returns the corresponding throughput and 95%-tail latency. Let $K = \{K_1, K_2, \cdots, K_n\}$ be the recommended configuration for each iteration, and $R = \{R_1, R_2, \cdots, R_n\}$ be the feedback obtained for each deployment configuration. We use Eq. 1 and Eq. 2 to fuse the two indicators in

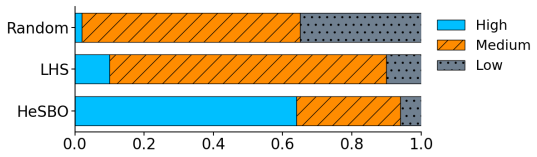


Figure 6: Comparison of efficiency in acquiring Redis database samples using different methods, where *High* and *Medium* respectively indicate a performance improvement of $> 30\%$ and $10\% - 30\%$ compared to the default, while *Low* indicates other situations.

K_t into R_t:

$$E(x) = \begin{cases} -10 \times (\frac{1}{e^x} - 1), x \leq 0 \\ 10 \times (e^x - 1), x > 0 \end{cases}, \tag{1}$$

$$f(K_t) = \beta E(\frac{T_{cur}}{T_{def}} - 1) + (1 - \beta)E(1 - \frac{L_{cur}}{L_{def}}), \tag{2}$$

where $E(x)$ is the amplification function which aims to enlarge the range of reward values. Here, T_{cur} and T_{def} are the throughput obtained by the current and the default configuration respectively, and L_{cur} and L_{def} are the 95%-tail latency obtained by the current and the default configuration respectively. $\beta \in [0, 1]$ is the weighted ratio of throughput and 95%-tail latency. The larger β, the more the algorithm favors throughput, otherwise it favors latency. β can be set by the user, here we set $\beta = 0.5$.

$S_t = \{K_t, R_t\}$ is the sample added for the t-th time in the observation set. In GP, every point in the continuous input space is associated with a normally distributed random variable. When the observation set changes from S_t to S_{t+1}, the GP model can be easily updated. Then EI is used to recommend the next optimal configuration, iterating until reaching the maximum iterations or performance improvement stagnates.

3.2 Knob Space Reduction

NoSQL databases have a very large configuration space. In order to reduce the input dimension of the model, we use the Random Forest (RF) [6] to calculate the feature importance of the configuration space. RF is a supervised machine learning method that uses multiple decision trees to classify or regress data, with the final output being determined by the mode of each tree's class.

We choose Classification and Regression Tree (CART) [28] to build the decision tree. The RF consists of 500 CART. First, G features are input, where features are a subset of configurations and the labels are their corresponding performance metrics. Next, it randomly collects n times with replacement from the 200 training samples obtained by BO to obtain a training dataset. And the undrawn samples are used as the test dataset. At each node, G features are randomly chosen for decision making, and CART constructs the tree based on these using Gini impurity [19]. Gini impurity represents the probability that a randomly selected sample is misclassified in the sample set. And the best features (i.e. knobs) are split from the nodes to generate new leaf nodes. Each CART will grow fully without pruning.

By averaging Gini impurity reduction across 500 CARTs, we determine knob importance. The input configuration space is filtered based on these values, yielding a ranked list. We choose the top-20 knobs as the input space for the reinforcement learning model.

3.3 Adaptability Improvement via Meta TD3

Most of existing database tuning methods based on reinforcement learning [29, 45, 50] use the Deep Deterministic Policy Gradient (DDPG)[30] algorithm. DDPG combines DQN and Actor-Critic, which is a model-free, off-policy method. However, the critic of DDPG tends to estimate a high Q-value, which is fatal in practice [46]. Therefore, we use the Twin Delayed Deep Deterministic Policy Gradient (TD3) algorithm, an improved version of DDPG [17]. TD3 mitigates overestimation issues by using two critics to estimate Q-values. It reduces the situation where the Q-value is too high by selecting the smallest one as the target Q-value.

Although TD3 can quickly learn tuning knowledge, it does not work well consistently for different environments. Existing work [29, 50] fine-tunes the model before tuning. However, the fine-tuning takes too long to quickly recommend an optimal configuration. Therefore, our work builds on meta-learning in the context of reinforcement learning [43], which adapts to unseen environment by aggregating historical experience into the latent representations. We use offline meta-reinforcement learning to solve tuning problems. We fuse meta-learning with TD3. The neural network is used to convert features under different workloads or hardware environments into contexts, which are used as input to *Actor*. It enhances the agent's ability to adapt to diverse environments rapidly. We explain the six model elements as follows.

Agent. *Agent* is the TD3 model. It comprises *Actor* and *Critic*. The *Actor* takes the current *State* and environment features as input, and outputs an *Action*. *Critic* takes the current *State*, *Action* and the features of the current environment as input, and outputs a Q-value. *Actor* updates based on *Critic*'s feedback, and *Critic* adjusts according to rewards. The *Agent*'s aim is to optimize *Action* within the tuning environment to maximize rewards and meet database performance requirements.

Environment. The *Environment* encompasses the running setup of the NoSQL databases and the OS environment. Meta-training uses multiple tuning environments on NoSQL databases for offline training. Different workloads, hardware and NoSQL databases make up different environments. In our implementation, we decouple the *Environment* from the tuning algorithm.

State. *State* is the current tuning system's observation of the current tuning environment. The *State* is divided into application states and OS states, which cover the state of CPU, memory, network, hard disk, file system and other categories. Each application has a different way to obtain the states. For example, MongoDB uses the "serverStatus" and "mongostat" commands, and Cassandra uses the "nodetool" command. The OS obtains the states through the "vmstat" command that comes with Linux.

Action. *Action* is the knobs for OS and NoSQL database, including the important knobs selected by the RF.

Reward. After the *Action* is converted into knobs and deployed in the actual environment, the current throughput and 95%-tail latency are obtained. Then they are substituted into Eq. 1 and Eq. 2 to get *Reward*, which describes the current performance compared to the default performance.

Policy. The neural network needs to learn the optimal *Policy* function, which generates *Action* according to the *Environment* feature and *State*. Its goal is to maximize *Reward*.

For offline meta-training, inspired by previous work [39], we designed **Multi-tasking Prioritized Replay Memory** for

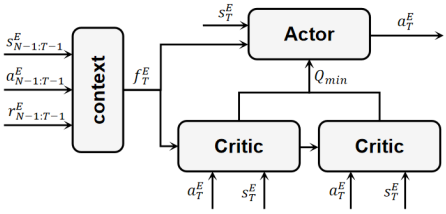


Figure 7: Meta TD3 algorithm process. To generate the value of a_T^E, the context from steps $N-1$ to $T-1$ is used.

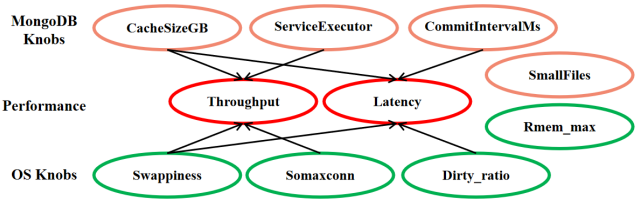


Figure 8: Partial ADMG of MongoDB. For the purpose of illustration, the causal relationships between 4 MongoDB knobs, 4 OS knobs and throughput and latency are shown.

experience replay. We store samples in the replay pool in the form of $\{E, P_T^E, s_T^E, s_{T+1}^E, a_T^E, r_T^E, s_{N:T}^E, a_{N:T}^E, r_{N:T}^E\}$, where E represents the tuning environment, s_T^E is the current *State*, a_T^E is the *Action* made by the *Actor* according to s_T^E and current *Environment* features f_T^E, s_{T+1}^E and r_T^E are the *State* and *Reward* of the *Environment* after performing a_T^E. $s_{N:T}^E, a_{N:T}^E, r_{N:T}^E$ are the *State*, *Action* and *Reward* of the E collected from steps N to T, where $N = T - 10$. P_T^E is the priority of the sample. It is calculated by Eq. 3 and Eq. 4.

$$\delta_T^E = |Q_1 - Q_2|, \tag{3}$$

$$P_T^E = \delta_T^E + |r_T^E|, \tag{4}$$

where Q_1 is the current Q function, and Q_2 is the target Q function. The absolute value of the difference between Q_1 and Q_2 is Temporal Difference (TD) error. If the TD error δ_T^E is large, it means that the current Q function is far from the target Q function, and more samples are required for training. In addition to focus on samples with a high TD error, samples with larger and smaller rewards also need to be considered. These values are sparse, but have a greater impact on the target Q value, providing crucial policy update information. Therefore, we consider TD error and *Reward* at the same time, so that samples with high TD error and high or low reward value can be used multiple times. The model can fully learn the knowledge of these high-value samples.

For Meta TD3, we create TD3-Context, TD3-Actor and TD3-Critic. TD3-Context uses a two-layer recurrent neural network GRU. Compared to LSTM, GRU has a simpler structure, fewer parameters, and easier convergence. TD3-Actor and TD3-Critic are implemented using simple neural networks. Similar to previous meta-RL [16, 37], we assume the tuning environment has a distribution $p(E)$, where each task is a Markov decision process (MDP) consisting of a set of *State*, *Action*, transition function, and reward function composition. The reward function is set in advance, and the transition function is learned through training. As shown in Fig. 7, given a set of different tuning environments $p(E)$, the meta-training process first converts the historical context c of an environment into the features, so as

Algorithm 1: Exploitation Based on Causal Inference

Input: High quality Sample set $\mathbb{S}$ in Replay Memory
Output: Configuration K generated by Exploitation strategy
Convert all actions in $\mathbb{S}$ to knobs and non-numeric values to numbers
/* Run FCI on $\mathbb{S}$ for causality discovery and get the causal paths list */
$Paths \leftarrow \mathbb{S}$
Draw ADMG according to *Paths*
Convert the knobs to discrete values and store them in a dictionary V
$KV \leftarrow$ empty list
for *Path in Paths* **do**
  for k *in Path* **do**
    $max\_effect \leftarrow -20000$
    for v *in* $V[k]$ **do**
      Calculate the causal effect cur_effect when the value of k is v according to the backdoor adjustment and ADMG
      if $max\_effect < cur\_effect$ **then**
        $max\_effect \leftarrow cur\_effect$
        Add $\{cur\_effect, k, v\}$ to KV
      end
    end
  end
end
Find the KV element with the highest cur_effect, set knob k to v, and use the best-performing values for other knobs in the current environment.
return Configuration K

to allow the policy to quickly adapt to unseen environments. Let $c_t^E = (s_t^E, a_t^E, r_t^E)$ be the context of environment E, where $t = 1 : T$. $c_{N:T}^E$ is the experience of environment E collected from steps N to T. Input $c_{N:T}^E$ into TD3-Context, and output the characteristic f_T^E of the environment. Then, TD3-Actor learns the tuning strategy through the strategy function. TD3-Critic evaluates the quality of the policy function through $Q(s, a, f)$.

In online tuning, deploying the trained model in varying environments necessitates rapid adaptation to previously unseen conditions. Initially, the *Collector* collects samples, which are then processed by the *Adapter*. The *Adapter* utilizes TD3-Context to extract environment features from the samples. These features, along with data from the Replay Memory, are inputted into a covariate shift correction model. This model computes the deviation between the current and training environments using a logistic algorithm, which has been shown to be effective in dealing with changes in the data distribution [41]. Finally, This model is used to adjust the offline model to obtain a model adapted to the current environment.

3.4 Exploitation Strategy with Causal Inference

To accelerate model convergence, we need to make full use of high-quality tuning samples. We focus on the trade-off dilemma in reinforcement learning between trying new *Action* (exploration) or choosing the best *Action* based on previous experience (exploitation). The existing methods involve changing the exploration strategy by adding random values to the best-performing *Action*, but this doesn't ensure continued optimal performance and can limit *Agent* exploration while risking local optima. In addition to efficient exploration, efficient exploitation can also hasten model convergence. Given limited training samples, effective sample utilization is critical for tuning. Finding the essential relationship between configuration and performance is pivotal for efficient sample usage. Inspired by Unicorn [23], we use causal inference [35] as an exploitation strategy. Causal inference infers causal relationships from data, which correlation-based methods cannot do [49]. It can help mine the essential relationship between knobs and performance.

Causal inference offers two advantages: (i) intervention, assessing impact changes given an intervention; (ii) explanation, understanding why a configuration can lead to high performance. It consists of two phases, the **information phase** and the **query phase**. The **information phase** uncovers causal structures, often leveraging conditional independence relationships in data. Common constraint-based algorithms like PC [25] and FCI [42] are used for this purpose. PC requires no confounding factors, while FCI can give nearly correct results even in the presence of confounding factors. During offline training, not all state factors are observable in the actual system, and accurate causal structures improve intervention accuracy. Therefore, we run the FCI algorithm based on the samples in Replay Memory for causal discovery. Next, we construct the Asyclic Directed Mixed Graph (ADMG) based on these causal relationships, illustrated for MongoDB in Fig. 8.

In the **query phase**, a specific knob is adjusted based on the determined causal structure, leveraging high-quality samples to expedite model convergence. Actions are generated according to ADMG. CTuner employs causal effects to identify and modify the NoSQL database configuration knob with the greatest performance impact. This process is detailed in Algorithm 1. Assuming all confounding factors are identifiable through observational data, backdoor adjustment is used to calculate the causal effect of knobs on performance, with the knob exhibiting the largest causal effect being adjusted. The **Exploitation Strategy Based on Causal Inference (ESCI)** is implemented using Causal-learn [4].

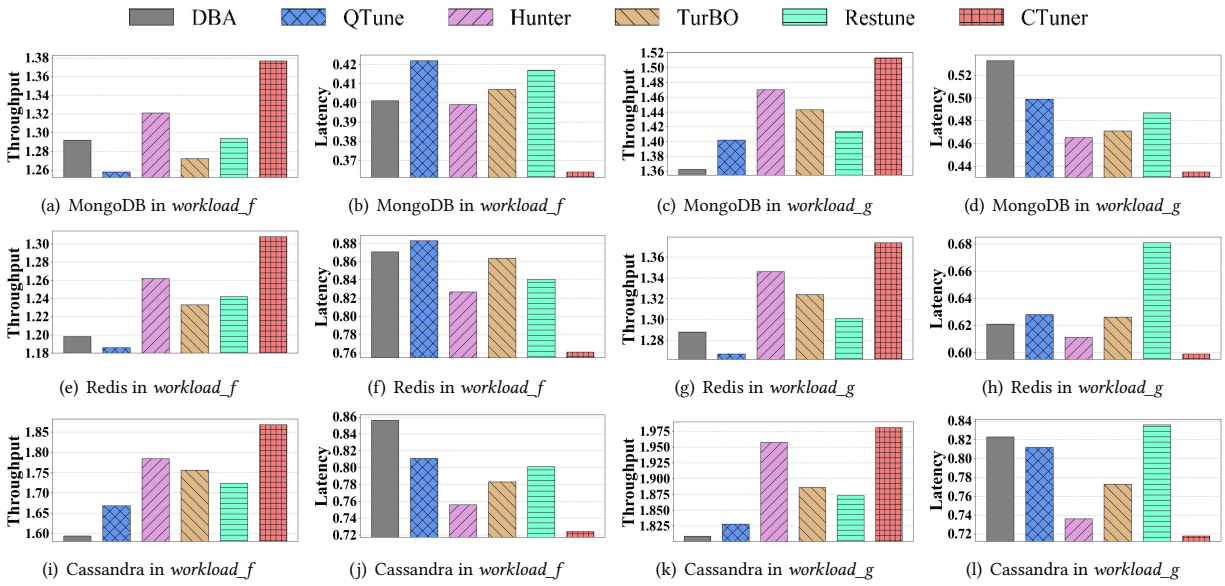
We incorporate causal inference into meta TD3. Below is a brief overview of the Action selection strategy. A small sample size can lead to low accuracy in causal discovery. Therefore, experiments determined that when the sample size in the Replay Memory exceeds 200, the causal inference function is activated. Once enabled, there's a 30% probability of using ESCI for *Action* selection. As the step of model training increases, the probability of using ESCI decreases, and eventually relying solely on the *Agent* for *Action* generation.

4 EXPERIMENTAL EVALUATION

We now present an evaluation on CTuner's ability to automatically optimize the configuration of a NoSQL database. We first implemented CTuner on the top of Pytorch [34], Sklearn [36] and Causal-learn [4]. Then, we compared CTuner with other advanced methods to illustrate its reliability and effectiveness. Finally, we conducted a series of ablation experiments to confirm the effectiveness of each module of CTuner.

4.1 Experiment Setup

System and environment. CTuner was evaluated on a LAN-connected ten-node NoSQL DB cluster, one CTuner node. The other 9 nodes are equally divided into 3 small clusters, each cluster is heterogeneous and deployed with Ubuntu 18.04 OS. Each node in two of the clusters is configured with 16 GB RAM and 8 cores (but the two clusters run on different CPU configurations). The other cluster is configured with 8GB RAM and 4 cores per node. We conducted tuning experiments on three NoSQL databases including MongoDB, Redis and Cassandra. Each is deployed in a distributed manner in a small cluster.



(a) MongoDB in *workload_f* (b) MongoDB in *workload_f* (c) MongoDB in *workload_g* (d) MongoDB in *workload_g*

(e) Redis in *workload_f* (f) Redis in *workload_f* (g) Redis in *workload_g* (h) Redis in *workload_g*

(i) Cassandra in *workload_f* (j) Cassandra in *workload_f* (k) Cassandra in *workload_g* (l) Cassandra in *workload_g*

Figure 9: Comparison of the best relative throughput and latency of CTuner, DBA, QTune, Hunter, TurBO, Restune for MongoDB, Redis, Cassandra in *workload_f* and *workload_g*.

Table 1: Knobs and State Metrics for NoSQL Database

	MongoDB	Redis	Cassandra
Initial Knobs	40	46	53
Filtered Knobs	13	15	17
State Metrics	23	14	11

Considered Knobs. Table 1 shows the number of *Initial Knobs* for each database instance based on expert experience, as well as the number of *Filtered Knobs* based on RF filtering. The knobs include both the OS and database knobs. The knobs selected by *Filtered Knobs* are mentioned in the tuning guide [1, 3, 5]. *State Metrics* is the number of *State* indicators of the NoSQL database, and the *State* also includes the database states and OS states. To mitigate the effects of performance variability caused by workload, network and other hardware, we repeated all experiments 3 times to increase the confidence of the results and the correctness of the conclusions.

Baseline. We compare CTuner with state-of-the-art methods on NoSQL databases, where methods include QTune [29] (DS-DDPG) and ResTune [52] (meta-learning), TurBO [14] (Bayesian optimization), Hunter [7] (hybrid method). For fairness, the initialization parameters of each method are randomized, and the Reply Memory is empty for each workload. In addition, a senior database administrator (DBA) was invited. We used the configuration given by the DBA in combination with the database tuning guide and personal experience for comparison.

Workload setup and replay. In the experiment, Ansible is used to automate the deployment of CTuner configuration. Specifically, the database is deployed to the local cluster through a playbook written in YAML, configurations are updated from the OS kernel and NoSQL database. The YCSB tool is used to run the benchmarks. We use the standard workload of YCSB for offline training and online tuning, using *workload_a* (50% read, 50% update), *workload_b* (95% read, 5% update), *workload_c* (100% read), *workload_d* (95% read, 5% insert), *workload_e* (95% scan, 5% insert), *workload_f* (50% read, 50% insert). In order to increase the richness of the workload, we have customized *workload_g* (40% read, 30% update, 30% insert). In our experiments, the weight ratio β of throughput and latency is set to 0.5 by default.

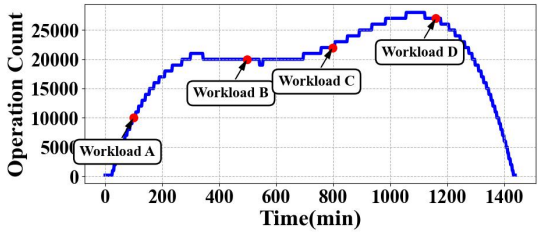


Figure 10: Workload changes over time.

4.2 Comparisons

To compare the performance of state-of-the-art methods, all methods start without any prior knowledge, and each method has the same offline training cost. The online tuning time also sets the step size to 100. We use *workload_a* to *workload_e* as the offline training environment, *workload_f* and *workload_g* as the online tuning environment. Each method is experimented with three NoSQLs, MongoDB, Redis, and Cassandra.

Fig. 9(a), 9(b) show the best relative throughput and best relative latency for each method on MongoDB for *workload_f*. As shown in Fig. 9(a), compared with DBA, QTune, Hunter, TurBO, and Restune, the best relative throughput of CTuner relative to the default performance is increased by 8.5%, 11.9%, 5.6%, 10.5%, and 8.3% for MongoDB in the same time cost. For latency, CTuner has the same advantage.

Fig. 9(e), 9(f), 9(i), 9(j) show the best relative throughput and best relative latency of each method on Redis, Cassandra for *workload_f*. According to the performance of these methods on *workload_f*, we can know that CTuner can increase the throughput by 4.6%-12.2% on Redis and 8.4%-27.4% on Cassandra compared with other

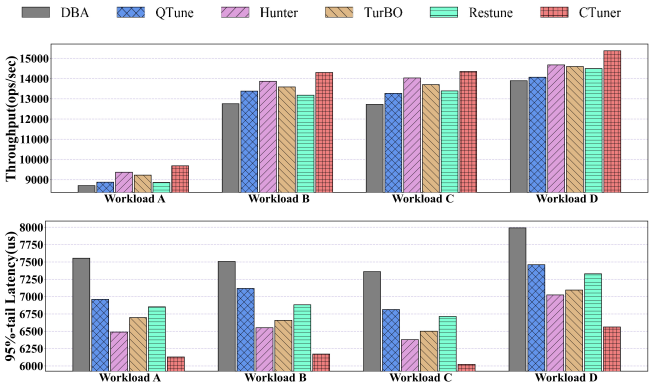


Figure 11: Comparison of throughput and latency of CTuner, DBA, QTune, Hunter, TurBO, Restune for MongoDB in dynamic workloads

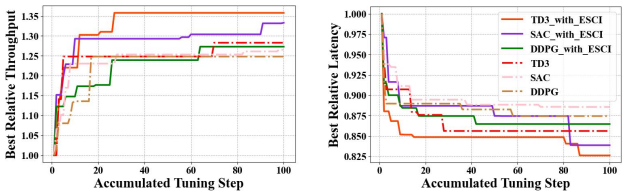


Figure 12: Comparison of the best relative throughput and latency of TD3, SAC, DDPG with and without ESCI for Redis.

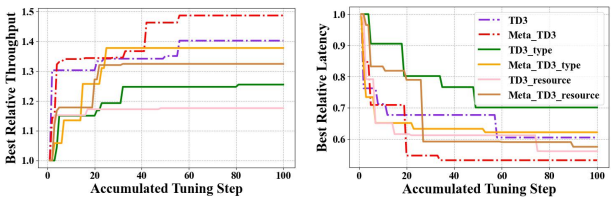


Figure 13: Comparison of the best relative throughput and latency of Meta TD3 and TD3 for MongoDB, where both are tuned in *workload_f*. Additionally, compare their performance on hardware with different CPU types (*TD3_type* and *Meta_TD3_type*) as well as on hardware with less CPU and memory resources (*TD3_resource* and *Meta_TD3_resource*).

methods. For latency, CTuner can decrease by 6.6%-12.2% on Redis and 3.2%-13.2% on Cassandra.

Fig. 9(c), 9(d), 9(g), 9(h), 9(k), 9(l) show the best relative throughput and performance of each method on MongoDB, Redis, Cassandra for *workload_g*. Each NoSQL database optimizes throughput and latency very differently in different workload. It shows that the environment has a great influence on the tuning system. However, CTuner still has advantages over other methods. CTuner can find a better configuration at the same time cost, with throughput increased by 2.4%-17.2% and 95%-tail latency decreased by 1.2%-11.8%. CTuner's meta-learning ability can quickly adapt to unseen environments and recommend the optimal configurations. In contrast, other methods require longer fine-tuning periods, leading to suboptimal recommendations within the same time constraints.

Real-world workloads typically exhibit diurnal patterns [38], with requests per second (RPS) varying throughout the day, such as higher RPS in the afternoon compared to the evening. To capture this variation, we use an e-commerce search benchmark that records the RPS of an e-commerce search system over a month [2]. This RPS value is scaled to obtain the *operationcount* in YCSB. The dynamic workload is derived from *workload_g*, with its *operationcount* varying according to the e-commerce benchmark. For performance comparison under dynamic workloads, we selected four time points for demonstration (Fig. 10). Results in a MongoDB cluster (Fig. 11) show that CTuner consistently outperforms other methods, demonstrating its ability to rapidly adjust configurations in response to workload changes, thereby ensuring superior NoSQL database performance.

4.3 Ablation Studies

Causal Inference and RL Module. Current RL-based database tuning methods primarily use DDPG. To validate our method, we experiment on TD3, SAC, and DDPG, both with and without ESCI, using the same training duration. Then, we use these offline trained models for online tuning. It can be seen from Fig. 12 that the method with ESCI can significantly improve the tuning effect. It shows that ESCI can effectively identifies causal relationships between knobs and enhances training efficiency. In addition, DDPG tends to overestimate Q-values early in training, whereas TD3 and SAC mitigate this with two *Critic* networks. TD3 and SAC outperform DDPG, with TD3 performing best in our experiments. Therefore, we choose TD3 with ESCI for the reinforcement learning model.

Meta-learning Module. We conduct experiments on the meta-learning module in MongoDB, and compare the methods of Meta TD3 and TD3, both using ESCI. We use *workload_a* to *workload_e* as the offline training environment. It can be seen from Fig. 13 that both methods have poor initial effects when encountering the unseen environment. However, meta TD3 can adapt to the environment through the *Adaptor* at the beginning. After adaptation, mata TD3 shows a better performance than TD3. This ability of rapid adaptation makes meta TD3 have better generalization ability and stronger applicability.

Table 2: The model inference time of each method for MongoDB in online learning (repeated 50 experiments, taking mean and variance)

	QTune	Hunter	Turbo	Restune	CTuner
Inference Time	4.6±0.4ms	4.4±0.5ms	152.1±21.5ms	436.5±46.8ms	5.0±0.5ms

Tuning Model Inference Time. Although the CTuner is more complex, its model inference time is still within an acceptable range. As shown in Table 2, CTuner's model inference time is similar to that of QTune and Hunter. In addition, RL-based methods have shorter inference times than BO-based methods. The time complexity of Bayesian optimization is $O(N^3)$ and incremental training cannot be carried out. As iterations increase, BO becomes slower [11]. During online tuning, configuration deployment and pressure testing times exceed 20 and 50 seconds, respectively. Therefore, CTuner's model inference time is acceptable.

5 LIMITATIONS

CTuner provides an effective, efficient and reliable configuration tuning method for NoSQL databases. However, it has some limitations. First, the reward function of TD3 is predefined. While it is

suitable for MongoDB, Redis and Cassandra, it is not necessarily suitable for other NoSQL databases. In the future, CTuner can be combined with inverse RL to allow the model to automatically learn the reward function. Secondly, we only use causal inference to improve the exploitation strategy of RL. Future work will allow causal inference and RL to have a deeper combination. Finally, CTuner's capability to quickly adapt to the unseen environment is achieved through a simple covariate shift correction model. Although it has a good effect at present, there is still a lot of space for improvement. Future work can explore how to better improve the adaptability of meta-learning models.

6 RELATED WORK

Database tuning. Currently, research on automatic database configuration tuning using machine learning can be divided into methods based on Bayesian Optimization (BO) and methods based on Reinforcement Learning (RL). The methods based on BO use previous observations to optimize iterations for fewer objective function evaluations and ease knob tuning effort. iTuned [15] uses statistics to determine which knobs are important based on their correlation with performance. It adopts BO for tuning for the first time. OtterTune [45] utilizes Lasso to identify key knobs and match new workloads with known ones. RelM [27] designed an experience-driven white-box algorithm to accelerate BO to provide memory allocation suggestions for distributed analysis engines. CGPTuner [8] adopts the context BO to adapt to workload changes. TurBO [14] combines adaptive pseudo-point mechanism with BO to reduce iterations and handles sub-optimal configurations based on ensemble learning and historical tuning experience. ResTune [52] adopts the method of BO, aiming to optimize the resource utilization, throughput and latency of the database system at the same time. Compared with these works, CTuner adopts RL for tuning, which can make full use of historical tuning experience and avoid the problem that BO-based methods tend to slow down the reasoning speed [11].

Reinforcement Learning. RL is a learning mechanism that learns how to map from state to action so as to maximize the reward obtained. Traditional RL methods have difficulty solving high-dimensional problems. Mnih et al. [31] proposed a DRL model that combines RL and deep learning to tackle the challenge of high-dimensional state and action spaces. CDBTune [50] is the first to use DRL for database tuning via DDPG algorithm. QTune [29] supports three tuning granularities and uses DDPG's actor-critic network to find the optimal configuration based on the current and predicted database state. Dremel [54] uses an online bandit model for configuration selection, supporting fast evaluation with multi-fidelity and upper-confidence-bound sampling. Hunter [7] is an online hybrid auto-tuning system for database. It combines meta-heuristic algorithm and DRL, and also designs cloning and parallelization scheme for fast tuning. Compared with these works, CTuner considers the knobs and states of the operating system, and designs causal inference algorithms to make full use of high-quality samples to accelerate model convergence.

Transfer Learning. It is crucial to models in different hardware or workload environments to maintain good performance. It leads to the concept of transfer learning, which involves fine-tuning or reusing pre-trained models for new tasks to accelerate training and improve performance. Three transfer techniques in configuration tuning, including *workload mapping*, *learning workload embedding*, and *model ensemble*, can enhance adaptability to dynamic workloads [55]. For *workload mapping*, OtterTune [45] and CGPTuner [8] create initial tuning models by identifying the most similar historical workloads. For *learning workload embedding*, Qtune [29] predicts database state changes based on workload characteristics such as operator types and query costs. WATuning [18] integrates reinforcement learning and attention neural networks to capture workload characteristics and accelerate convergence for new workloads. For *model ensemble*, ResTune [52] employs an ensemble framework to combine historical knowledge, generating a meta-learner for the target workload. DeepCAT [13] adapts to new environments using knowledge transfer techniques based on Progressive Neural Networks (PNN). Compared with these works, CTuner innovatively combines reinforcement learning with meta-learning for model transfer, leveraging historical experience to abstract common patterns, thereby enabling rapid adaptation to unseen environments.

7 CONCLUSION

In this paper, we propose a hybrid tuning system for NoSQL databases named CTuner, which is designed to achieve efficient and accurate configuration search. Initially, CTuner generates high-quality training samples through HeSBO. Subsequently, we use the random forest to filter these samples to keep only important knobs, thereby reducing the complexity of the search space. Moreover, we introduce causal inference into the improvement process of reinforcement learning strategies to search more efficiently in limited training samples. Additionally, we combine meta-learning and RL methods to enable CTuner to adapt to unseen environments more quickly. To verify the performance of CTuner, we conduct experiments using YCSB as a benchmark. Compared with the advanced system, CTuner can find a better configuration at the same time cost with throughput increase by 2.4%-27.4% and 95%-tail latency decrease by 1.2%-13.2%.

ACKNOWLEDGMENTS

The research is supported by Key Area Research and Development Program of Guangdong Province (No.2020B010165002), the National Natural Science Foundation of China (No.62272495), the Basic and Applied Basic Research of Guangzhou (No.202002030328), the Guangdong Basic and Applied Basic Research Foundation (No. 2023B1515020054). The corresponding author is Pengfei Chen.

REFERENCES

[1] 2016. Cassandra partial knob tuning guide. https://www.slideshare.net/DataStax/a-detailed-look-at-cassandrayaml-edward-capriolo-the-last-pickle-cassandra-summit-2016.
[2] 2020. E-commerce search benchmark. https://github.com/alibaba/eCommerceSearchBench.
[3] 2021. MongoDB partial knob tuning guide. https://www.percona.com/blog/mongodb-101-5-configuration-options-that-impact-performance-and-how-to-set-them.
[4] 2022. Causal-learn: Causal Discovery for Python. https://github.com/cmu-phil/causal-learn.
[5] 2022. Redis partial knob tuning guide. https://devpress.csdn.net/linux/62eba2d9648466712833a7c8.html.
[6] Leo Breiman. 2001. Random forests. *Machine learning* 45, 1 (2001), 5–32.
[7] Baoqing Cai, Yu Liu, Ce Zhang, Guangyu Zhang, Ke Zhou, Li Liu, Chunhua Li, Bin Cheng, Jie Yang, and Jiashu Xing. 2022. HUNTER: An Online Cloud Database

Hybrid Tuning System for Personalized Requirements. In *Proceedings of the 2022 International Conference on Management of Data*. 646–659.
[8] Stefano Cereda, Stefano Valladares, Paolo Cremonesi, and Stefano Doni. 2021. CGPTuner: a contextual gaussian process bandit approach for the automatic tuning of IT configurations under varying workload conditions. *Proceedings of the VLDB Endowment* 14, 8 (2021), 1401–1413.
[9] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078* (2014).
[10] Brian F Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. 2010. Benchmarking cloud serving systems with YCSB. In *Proceedings of the 1st ACM symposium on Cloud computing*. 143–154.
[11] Valentin Dalibard. 2017. *A framework to build bespoke auto-tuners with structured Bayesian optimisation*. Technical Report. University of Cambridge, Computer Laboratory.
[12] 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.
[13] Hui Dou, Yilun Wang, Yiwen Zhang, and Pengfei Chen. 2022. DeepCAT: A Cost-Efficient Online Configuration Auto-Tuning Approach for Big Data Frameworks. In *Proceedings of the 51st International Conference on Parallel Processing*. 1–11.
[14] Hui Dou, Lei Zhang, Yiwen Zhang, Pengfei Chen, and Zibin Zheng. 2023. TurBO: A cost-efficient configuration-based auto-tuning approach for cluster-based big data frameworks. *J. Parallel and Distrib. Comput.* 177 (2023), 89–105.
[15] Songyun Duan, Vamsidhar Thummala, and Shivnath Babu. 2009. Tuning database configuration parameters with ituned. *Proceedings of the VLDB Endowment* 2, 1 (2009), 1246–1257.
[16] Rasool Fakoor, Pratik Chaudhari, Stefano Soatto, and Alexander J Smola. 2019. Meta-q-learning. *arXiv preprint arXiv:1910.00125* (2019).
[17] 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.
[18] 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.
[19] Corrado Gini. 1912. *Variabilità e mutabilità: contributo allo studio delle distribuzioni e delle relazioni statistiche.[Fasc. I.]*. Tipogr. di P. Cuppini.
[20] Yaniv Gur, Dongsheng Yang, Frederik Stalschus, and Berthold Reinwald. 2021. Adaptive Multi-Model Reinforcement Learning for Online Database Tuning.. In *EDBT*. 439–444.
[21] Shiyue Huang, Yanzhao Qin, Xinyi Zhang, Yaofeng Tu, Zhongliang Li, and Bin Cui. 2023. Survey on performance optimization for database systems. *Science China Information Sciences* 66, 2 (2023), 1–23.
[22] Ronald L Iman, James M Davenport, and Diane K Zeigler. 1980. *Latin hypercube sampling (program user's guide).[LHC, in FORTRAN]*. Technical Report. Sandia Labs., Albuquerque, NM (USA).
[23] Md Shahriar Iqbal, Rahul Krishna, Mohammad Ali Javidian, Baishakhi Ray, and Pooyan Jamshidi. 2022. Unicorn: reasoning about configurable system performance through the lens of causality. In *Proceedings of the Seventeenth European Conference on Computer Systems*. 199–217.
[24] Yoshiteru Ishihara and Masahito Shiba. 2020. Dynamic Configuration Tuning of Working Database Management Systems. In *2020 IEEE 2nd Global Conference on Life Sciences and Technologies (LifeTech)*. IEEE, 393–397.
[25] Markus Kalisch and Peter Bühlman. 2007. Estimating high-dimensional directed acyclic graphs with the PC-algorithm. *Journal of Machine Learning Research* 8, 3 (2007).
[26] Konstantinos Kanellis, Cong Ding, Brian Kroth, Andreas Müller, Carlo Curino, and Shivaram Venkataraman. 2022. LlamaTune: Sample-Efficient DBMS Configuration Tuning. *arXiv preprint arXiv:2203.05128* (2022).
[27] Mayuresh Kunjir and Shivnath Babu. 2020. Black or White? How to develop an autotuner for memory-based analytics. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 1667–1683.
[28] Roger J Lewis. 2000. An introduction to classification and regression tree (CART) analysis. In *Annual meeting of the society for academic emergency medicine in San Francisco, California*, Vol. 14. Citeseer.
[29] 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.
[30] 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).
[31] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. 2015. Human-level control through deep reinforcement learning. *nature* 518, 7540 (2015), 529–533.
[32] CKJDSA Stefan Mühlbauer, Florian Sattler, and N Siegmund. 2023. Analyzing the impact of workloads on modeling the performance of configurable software systems. In *Proceedings of the International Conference on Software Engineering (ICSE), IEEE*.
[33] Amin Nayebi, Alexander Munteanu, and Matthias Poloczek. 2019. A framework for Bayesian optimization in embedded subspaces. In *International Conference on Machine Learning*. PMLR, 4752–4761.
[34] 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).
[35] Judea Pearl. 2009. *Causality*. Cambridge university press.
[36] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.
[37] Kate Rakelly, Aurick Zhou, Chelsea Finn, Sergey Levine, and Deirdre Quillen. 2019. Efficient off-policy meta-reinforcement learning via probabilistic context variables. In *International conference on machine learning*. PMLR, 5331–5340.
[38] Charles Reiss, Alexey Tumanov, Gregory R Ganger, Randy H Katz, and Michael A Kozuch. 2012. Heterogeneity and dynamicity of clouds at scale: Google trace analysis. In *Proceedings of the third ACM symposium on cloud computing*. 1–13.
[39] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. 2015. Prioritized experience replay. *arXiv preprint arXiv:1511.05952* (2015).
[40] Rami Sellami and Bruno Defude. 2017. Complex queries optimization and evaluation over relational and NoSQL data stores in cloud environments. *IEEE transactions on big data* 4, 2 (2017), 217–230.
[41] Hidetoshi Shimodaira. 2000. Improving predictive inference under covariate shift by weighting the log-likelihood function. *Journal of statistical planning and inference* 90, 2 (2000), 227–244.
[42] Peter Spirtes, Clark N Glymour, Richard Scheines, and David Heckerman. 2000. *Causation, prediction, and search*. MIT press.
[43] Bradly C Stadie, Ge Yang, Rein Houthooft, Xi Chen, Yan Duan, Yuhuai Wu, Pieter Abbeel, and Ilya Sutskever. 2018. Some considerations on learning to explore via meta-reinforcement learning. *arXiv preprint arXiv:1803.01118* (2018).
[44] Pavel Valov, Jean-Christophe Petkovich, Jianmei Guo, Sebastian Fischmeister, and Krzysztof Czarnecki. 2017. Transferring performance prediction models across different hardware platforms. In *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering*. 39–50.
[45] 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.
[46] Hado Van Hasselt, Arthur Guez, and David Silver. 2016. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 30.
[47] Edward Wagstaff, Fabian Fuchs, Martin Engelcke, Ingmar Posner, and Michael A Osborne. 2019. On the limitations of representing functions on sets. In *International Conference on Machine Learning*. PMLR, 6487–6494.
[48] Junxiong Wang, Immanuel Trummer, and Debabrota Basu. 2021. UDO: universal database optimization using reinforcement learning. *arXiv preprint arXiv:2104.01744* (2021).
[49] Shuai Yang, Hao Wang, Kui Yu, Fuyuan Cao, and Xindong Wu. 2021. Towards efficient local causal structure learning. *IEEE Transactions on Big Data* 8, 6 (2021), 1592–1609.
[50] 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.
[51] Xinyi Zhang, Zhuo Chang, Hong Wu, Yang Li, Jia Chen, Jian Tan, Feifei Li, and Bin Cui. 2023. A Unified and Efficient Coordinating Framework for Autonomous DBMS Tuning. *Proceedings of the ACM on Management of Data* 1, 2 (2023), 1–26.
[52] 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.
[53] Xinyi Zhang, Hong Wu, Yang Li, Jian Tan, Feifei Li, and Bin Cui. 2022. Towards dynamic and safe configuration tuning for cloud databases. In *Proceedings of the 2022 International Conference on Management of Data*. 631–645.
[54] Chenxingyu Zhao, Tapan Chugh, Jaehong Min, Ming Liu, and Arvind Krishnamurthy. 2022. Dremel: Adaptive Configuration Tuning of RocksDB KV-Store. In *Abstract Proceedings of the 2022 ACM SIGMETRICS/IFIP PERFORMANCE Joint International Conference on Measurement and Modeling of Computer Systems*. 61–62.
[55] Xinyang Zhao, Xuanhe Zhou, and Guoliang Li. 2023. Automatic Database Knob Tuning: A Survey. *IEEE Transactions on Knowledge and Data Engineering* (2023).