XTuning: Expert Database Tuning System Based on Reinforcement Learning

Yanfeng Chai[1,2], Jiake Ge[1], Yunpeng Chai[1(✉)], Xin Wang[3], and BoXuan Zhao[1]

[1] Key Laboratory of Data Engineering and Knowledge Engineering, MOE, and School of Information, Renmin University of China, Beijing, China
{yfchai,gejiake,ypchai,boxuan.zhao}@ruc.edu.cn
[2] College of Computer Science and Technology, Taiyuan University of Science and Technology, Taiyuan, Shanxi, China
[3] College of Intelligence and Computing, Tianjin University, Tianjin, China
wangx@tju.edu.cn

Abstract. Database performance optimization has become a hot issue in recent years. Some works deeply reconstruct the database to achieve specified goals like throughput or latency. The others focus on the database's configuration knobs with reinforcement learning (RL) to improve the performance without any empirical knowledge. But the exhaustive offline training process costs plenty of time and resources due to the large inefficient configuration knobs combinations with trial-and-error methods. The most time-consuming part of the process is not the RL network training, but the database performance evaluation for acquiring the reward values of target performance like throughput or latency. So we propose an expert database tuning system (XTuning) which contains a correlation knowledge model to remove unnecessary training costs and a multi-instance mechanism (MIM) to support fine-grained tuning for diverse workloads. The models define the importance and correlations among these configuration knobs for the user's specified target. Then we implement the models as Progressive Expert Knowledge Tuning (PEKT) algorithm with an abstracted architectural optimization integrated into XTuning. Experiments show that XTuning can effectively reduce the training time and achieves extra performance promotion compared with the state-of-the-art tuning methods.

Keywords: Database optimization · Auto tuning · Expert knowledge rules · Reinforcement learning · Reduce training time

1 Introduction

Databases usually offer a larger number of configuration knobs for users. For example, MySQL and PostgreSQL have about 200+ knobs while the key-value

This work is supported by the National Key Research and Development Program of China (No. 2019YFE0198600), National Natural Science Foundation of China (No. 61972402, 61972275, and 61732014).

W. Zhang et al. (Eds.): WISE 2021, LNCS 13080, pp. 101–110, 2021.
https://doi.org/10.1007/978-3-030-90888-1_8

store RocksDB [7] still has more than 100+ knobs for performance tuning. Therefore, tuning the hundreds of knobs is an impossible mission even for the very experienced DBAs. In another word, finding the optimal knobs solution in the huge continuous space is an NP-hard problem [14]. Existing knobs-based auto-tuning methods usually have some weaknesses during the training process. First, there is no fine-grained tuning mechanism for a specified workload. Second, plenty of time (dozens of hours to days) [16] and resources are spent on the offline performance measurements, which include many invalid knobs combinations lacking in correlation and feasibility check. In fact, these costs could be avoided with the constraint rules based on existing experiences or knowledge. The contributions are summarized as follows:

(1) We propose an expert database tuning system (XTuning) based on reinforcement learning, which contains the correlation rules module based on expert knowledge for different scenarios.
(2) We extend XTuning with Progressive Expert Knowledge Tuning (PEKT) algorithm included an abstraction method of the architectural optimization and multi-instance mechanism (MIM) for further performance promotion.
(3) Experiments show XTuning outperforms the SOTA auto-tuning method CDBTune both on training time reduction and on performance promotion.

2 Related Work

We summarize existing works related to the database auto-tuning as follows:

Knobs-based Auto-tuning. BestConfig [17] uses search-based methods to find the optimal knobs based on historical tuning data, which costs lots of time and needs to restart the process if a new request arrives. OtterTune [14] uses a learning-based method to recommend knobs based on the historical tuning experience. But this method requires higher quality samples with every necessary condition. CDBTune [16] adopts deep reinforcement learning (DRL) with a deep deterministic policy gradient method and a try-and-error strategy to find the optimal knobs through many performance tests. QTune [10] also utilizes a DRL model, focusing on the SQL's pattern for fine-grained tuning at a different level.

Knobs-based optimization will not be the final destination for auto-tuning. With key-value stores emerging, databases like to use it as the storage engine like Snowflake [3] and MyRocks [11]. KVStores are widely used in distributed system, such as TiDB [8], CockroachDB [13], NebulaGraph [12], and HugeGraph [9].

Architectural Optimization. SILK [1] designs an I/O scheduler to deal with the latency spikes by dynamically allocating I/O resources according to the operations' priorities. ALDC [2] focuses on LSM-tree's compaction mechanism with controllable granularity methods to acquire specified goals. Monkey [5] focuses on the bloom filter for performance promotion. Dostoevsky [6] further proposed a hybrid merge policy to remove superfluous merging adaptively. Bourbon [4] uses machine learning to build a learned index to promote the lookup performance.

3 Expert Knowledge-Based Tuning Architecture

Figure 1 presents the overall architecture of XTuning, and the gray arrow represents the training process. First, the external expert rules module classifies workloads from the system module. Then it (MIM) passes the classified workload to the corresponding core tuning algorithm. It could support fine-grained workloads classification methods under different scenarios while CDBTune only supports coarse-grained way. Second, the auto-tuning module receives the classified workload and other parameters, then trains itself efficiently with internal expert rules. They could significantly reduce the RL network training times and enhance the practicability while others like CDBTune cost too long. Finally, the auto-tuning module recommends knobs to the system. The above steps are repeated until the RL algorithm converges. RL utilizes a try-and-error strategy to explore optimal knobs' combinations, normally ignoring some configuration knobs that naturally have some kind of positive or negative correlations.

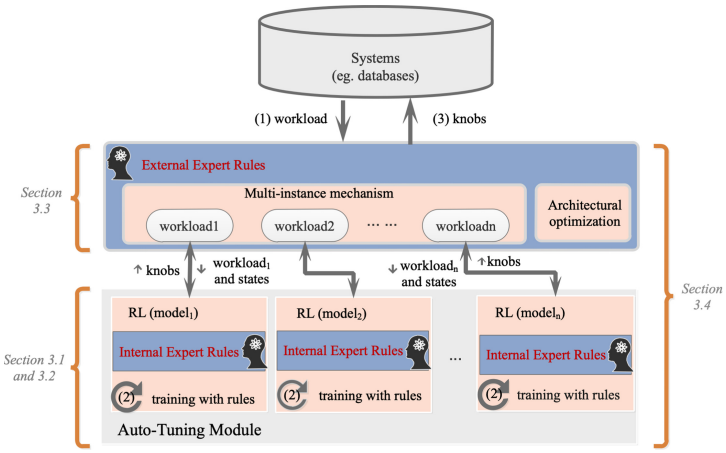


Fig. 1. System architecture of XTuning.

3.1 Correlation Rules Table of Knobs

As Table 1 shown, the architecture of correlation rules in XTuning is something like a two-dimensional table. XTuning utilizes it to offer a fine-grained tuning optimization under diverse workloads. $r_{i \to j(\cdot)}$ means the correlation between knob_i and knob_j with positive $(+)$, negative $(-)$, or uncorrelated (ϕ).

For example, LevelDB provides two knobs for the in-memory write buffer size and max file size. The buffer will be serialized into the disk as one or more files. Now we assume that we already know this expert knowledge, which means the $knob_{write\_buffer\_size}$ and the $knob_{max\_file\_size}$ should conform to positive correlation and $knob_{write\_buffer\_size} \geq 2.0 \cdot knob_{max\_file\_size}$. The correlation can be presented as $(+)2.0$, which leads to the least space-amplification for file systems. But if the two knobs have the wrong correlation, this meaningless performance testing will be skipped to reduce training time.

Table 1. Correlation rules table of knobs.

Workload_i	knob_1	knob_2	$\cdots$	knob_{n-1}	knob_n
knob_1	-	-		-	-
knob_2	$r_{2\to1(+)}$	-		-	-
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
knob_{n-1}	$r_{n-1\to1(-)}$	$r_{n-1\to2(-)}$		-	-
knob_n	$r_{n\to1(+)}$	$r_{n\to2(-)}$		$r_{n\to n-1(\phi)}$	-

3.2 Knobs Correlation with Internal Expert Knowledge

In Fig. 2, the red part represents the performance testing process, which provides the reward values to the RL network. To solve this problem, we embed an internal expert rules mechanism. Figure 2 shows that (1) If the RL outputs a knob that conforms to the internal expert rules, it seems to be a high-performance knob from an experiential perspective. The reward value is determined by the performance test module (red lines). (2) If the RL outputs a knob that does not conform to the internal expert rules, it seems to be a low-performance knob from an experiential perspective. So the real test is not needed, and it will be replaced by the experience reward (gray lines) for the system's availability.

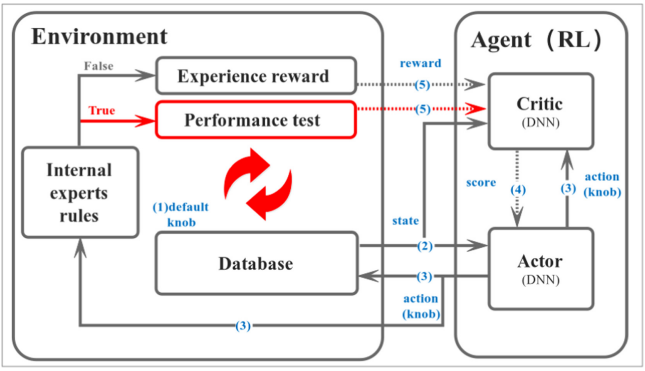


Fig. 2. Workflows of the auto-tuning module. (Color figure online)

3.3 Workloads Correlation with External Expert Knowledge

The workloads in real scenarios change dynamically. Existing auto-tuning methods like CDBTune cannot deal with these situations. In this part, we will describe how the external expert rules work in XTuning.

Multi-instance Mechanism (MIM). In Table 2, the external expert rules implement the classification based on the read/write ratio. XTuning trains each network individually based on these workloads. For example, MIM is monitoring

the real-time status of the workload. Once it reaches the threshold, MIM will re-select the auto-tuning modules to serve according to the next workload's pattern, and recommend high-performance knobs again. Note that (1) the internal structure of the input and output and the auto-tuning module are fixed. Therefore, XTuning only needs to establish a general neural network framework to load the internal parameters of the network corresponding to different instance models rather than to reestablish the neural network. (2) When XTuning training multiple models, the workload proportion generated by the process can meet the random value for the corresponding model's range. (3) MIM can generate the specified number of models according to the user's demand. XTuning could dynamically recommend the optimal knobs which fit the current workload's status. Therefore, comparing to CDBTune's coarse-grained model, MIM could classify workloads with a fine-grained method for better tuning.

Table 2. Multi-instance mechanism for the fine-grained tuning.

Write (%)	$(0, A_1]$	$(A_1, A_2]$	$(A_2, A_3]$		$(A_{i-1}, A_i]$
Instance	$model_1$	$model_2$	$model_3$		$model_i$
Read : Scan	$a_1 : b_1$	$a_2 : b_2$	$a_3 : b_3$	.....	$a_i : b_i$
Threshold	Th_1	Th_2	Th_3		Th_i

Abstract Architectural Optimization as Extra Knobs. As we mentioned in Sect. 1, We import a Fine-grained Controllable Compaction (FCC) mechanism in LevelDB, which further improves the performance and system fluctuation by controlling the write amplification (WA) for different read/write ratio workloads. As shown in Eq. 1, R_m means the compaction ratio in Fine-grained Controllable Compaction (FCC) mechanism. $\sum_{i=1}^{n} S_i$ is the current total size accumulated and F_{max} is the max file size. If the number exceeds N_{max} or $R_m \geq R_{th}$, then the compaction will be triggered immediately. So the ratio threshold R_{th} can control the compaction granularity to acquire specified performance. Therefore, we abstract the ratio as an expert knob in PEKT to achieve further performance promotion.

$$R_m = \begin{cases} (\sum_{i=1}^{n} S_i)/F_{max} & , i < N_{max}, \\ R_{th} & , otherwise. \end{cases} \tag{1}$$

3.4 Progressive Expert Knowledge Tuning Algorithm

Next, we integrate the FCC mechanism, the internal and external expert rules into XTuning as a progressive expert knowledge tuning algorithm (PEKT).

Algorithm 1. Progressive Expert Knowledge Tuning: Training Phase

Input: workloads sequence $W_n\{W_1, W_2, \cdots, W_\tau\}$

1: $model_i \leftarrow external\ expert\ rules(W_n)$
2: **In each episode:**
3: $critical\ params \leftarrow Database(W_n)$
4: $RL_i \leftarrow model_i$
5: **for** each time step β **do**
6: $knob \leftarrow RL_i\_actor(params)$
7: **if** $internal\ expert\ rules(knob) == True$ **or** $random() < \epsilon$ **then**
8: $reward_{RL_i} \leftarrow Eval(knob)$
9: **else**
10: $reward_{RL_i} \leftarrow Exp.\ reward$
11: **end if**
12: $train\ RL_i\_critic(reward\_RL)$
13: $train\ RL_i\_actor(score)$
14: **end for**

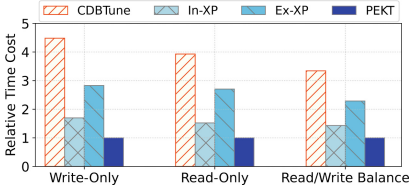
External Expert Rules. (1) In Algorithm 1, the external expert rules invoke a multi-instance mechanism to identify and classify the current workload. At the same time, the multi-instance mechanism forces the auto-tuning module to establish the corresponding RL network instance (Line 4). (2) In Algorithm 2, when XTuning receives the tuning signal and current operations' type, the external expert rules invoke the multi-instance mechanism to identify the current workload. Then the external expert rules inform the auto-tuning module to load the corresponding RL network instance (Line 3) for auto-tuning (Line 4).

Algorithm 2. Progressive Expert Knowledge Tuning: Running Phase

Input: workloads sequence $W_n\{W_1, W_2, \cdots, W_\tau\}$

1: $model_i \leftarrow external\ expert\ rules(W_n)$
2: $important\ params \leftarrow Database(W_n)$
3: $RL_i \leftarrow model_i$
4: $knob \leftarrow RL_i\_actor(params)$

Internal Expert Rules. (1) During the training phase in Algorithm 1, internal expert rules directly participate in and simplify the RL training process. If the knob does not conform to the internal expert rules, the performance testing phase will be skipped (Line 8). But there is still a tiny probability $P(r < \epsilon)$ to have a random exploration for rules' self-improvement. Rewards are set based on the experience to reduce the training time (Line 10); (2) Internal expert rules only serve the training period of XTuning, rather than the actual tuning period.



	WO	RO	RWB
CDBTune	103h	106h	117h
In-XP	39h	41h	50h
Ex-XP	65h	73h	80h
PEKT	23h	27h	35h

Fig. 3. Comparison of training time cost with CDBTune, In-XP, Ex-XP and PEKT.

4 Experiment Study

We implement XTuning based on the CDBTune [16]. Our evaluation is based on YCSB [15] and LevelDB for architectural optimization. We generate 5 GB of data and 50 M operations with 5 threads for each testing round. Each key-value pair is set to have a 16-B key and a 1-KB value.

4.1 Training Time Reduction with Expert Rules

We evaluate the different modules in XTuning with the Internal Expert Rules (In-XP), the External Expert Rules (Ex-XP), and Progressive Expert Knowledge Tuning (PEKT) with full features. Then we make a comparison of offline training time reduction with the above groups under three workloads (*Write-Only*, *Read-Only*, and *Read/Write-Balance*). First, all three can effectively reduce the offline training time in Fig. 3. Moreover, PEKT can reduce the offline training time by 77.67%, 74.53%, and 70.09% under WO, RO, and RWB workloads. Second, internal expert rules could reduce the performance testing cost with the correlation rules to accelerate RL network training. That means In-XP still achieves time reductions by 62.14%, 61.32%, and 57.26% under the above three workloads. Third, due to the external expert rules (Sect. 3.3), Ex-XP still outperforms CDBTune with time reductions by 36.89%, 31.13%, and 31.62%.

4.2 Throughput Improvement

In Fig. 4, PEKT can achieve the best performance promotion under different read/write ratios workloads. Compared with the default configuration setting, PEKT can promote the throughput by 3.8x $\sim$ 10.02x. Meanwhile, PEKT also can achieve about 4.58% $\sim$ 64.26% higher throughput improvement than CDBTune.

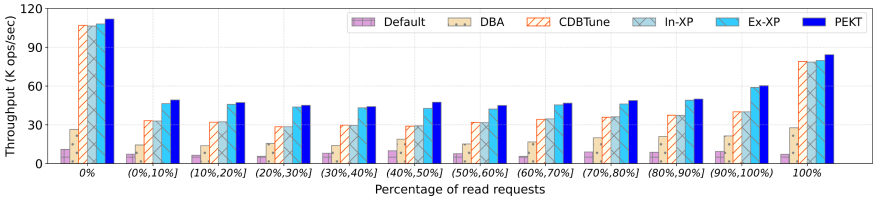


Fig. 4. Throughput comparison under the different read/write ratios workloads.

However, PEKT merely gets a tiny superiority under RO and WO workloads compared with the CDBTune. Because CDBTune focuses only on the two workloads, but PEKT utilizes the MIM for fine-grained read/write ratios workloads. Moreover, architectural optimization in XTuning further improves the throughput due to the FCC mechanism for controllable write amplification.

4.3 Latency Reduction

In this part, we evaluate the latency reduction under different read/write ratios workloads. We know the fluctuation usually ruins users' experience due to the terrible tail latency. In Sect. 3.4, we abstracted the FFC as an extra knob to import architectural optimization into XTuning. So PEKT could effectively reduce the tail latency by 53.88% $\sim$ 94.39% and 23.47% $\sim$ 63.45% compared with the Default and the CDBTune. Due to the FCC, PEKT can restrict latency into a more reasonable range to offer a smooth user experience (Fig. 5).

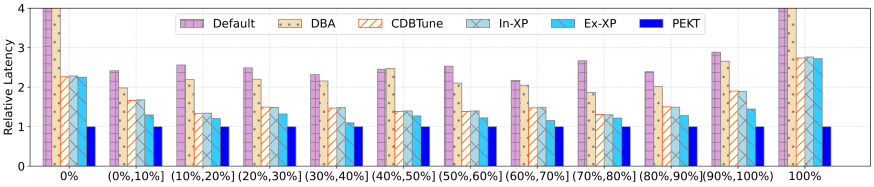


Fig. 5. Latency comparison under the different read/write ratios workloads.

4.4 Architectural Optimization Performance in XTuning

First, in Fig. 6(a), PEKT reduces internal I/O size by 52.9% and 19.02% compared with the Default and the CDBTune under *RO* workload. Second, PEKT reduces I/O size by 75.04% and 24.74% under WR workload in Fig. 6(d). Third, FCC effectively reduces the compaction I/O size by 80.21% and 54.01% compared with the Default and the CDBTune under WO workload in Fig. 6(g).

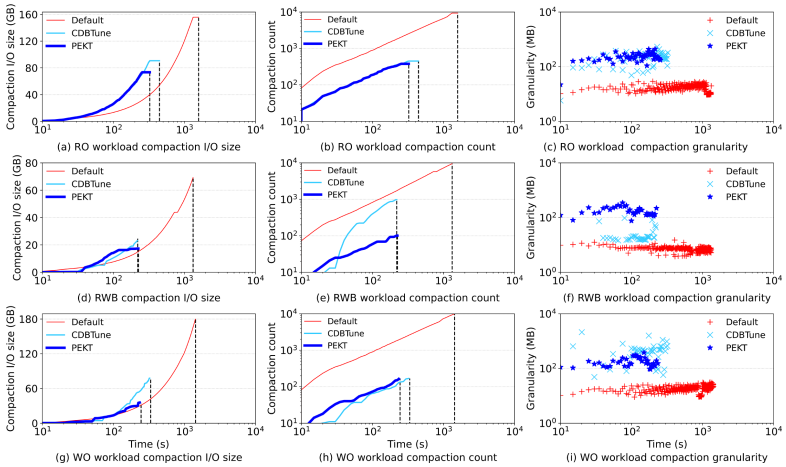


Fig. 6. Internal compaction I/O size, count and granularity under different workloads.

Figure 6(c), (f), and (i) describe the internal I/O granularity under *RO*, *RWB*, and *WO* workloads. The Default has the smallest compaction granularity because of its configuration space containing no optimizations for the workloads. Though the CDBTune can achieve close throughput performance with PEKT under *RO* and *WO* workloads, it still cannot control the compaction granularity, which may lead to terrible system fluctuations.

5 Conclusion

Existing auto-tuning methods usually ignore the correlations between the knobs and the workloads. Therefore, we propose XTuning with the internal and external expert knowledge modules to skip the unnecessary training rounds for reducing the training time with a fine-grained tuning for the complex workloads. Moreover, we integrate the architectural optimization into the XTuning, which leads to further performance promotion with the specified demands.

References

1. Balmau, O., Dinu, F., Zwaenepoel, W., Gupta, K., Chandhiramoorthi, R., Didona, D.: SILK: preventing latency spikes in log-structured merge key-value stores. In: 2019 USENIX Annual Technical Conference, pp. 753–766, July 2019
2. Chai, Y., Chai, Y., Wang, X., Wei, H., Wang, Y.: Adaptive lower-level driven compaction to optimize LSM-tree key-value stores. IEEE Trans. Knowl. Data Eng. (2020, early access). https://doi.org/10.1109/TKDE.2020.3019264
3. Dageville, B., et al.: The snowflake elastic data warehouse. In: Proceedings of the 2016 International Conference on Management of Data, pp. 215–226 (2016)
4. Dai, Y., et al.: From wiscKey to bourbon: a learned index for log-structured merge trees. In: 14th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 20), pp. 155–171 (2020)
5. Dayan, N., Athanassoulis, M., Idreos, S.: Monkey: optimal navigable key-value store. In: SIGMOD, pp. 79–94. ACM (2017)
6. Dayan, N., Idreos, S.: Dostoevsky: better space-time trade-offs for LSM-tree based key-value stores via adaptive removal of superfluous merging. In: Proceedings of the 2018 International Conference on Management of Data, pp. 505–520 (2018)
7. Dong, S., Callaghan, M., Galanis, L., Borthakur, D., Savor, T., Strum, M.: Optimizing space amplification in RocksDB. In: CIDR (2017)
8. Huang, D., et al.: TiDB: a raft-based HTAP database. Proc. VLDB Endowment **13**(12), 3072–3084 (2020)
9. Hugegraph (2021). https://github.com/hugegraph/hugegraph
10. Li, G., Zhou, X., Li, S., Gao, B.: Qtune: a query-aware database tuning system with deep reinforcement learning. Proc. VLDB Endowment **12**(12), 2118–2130 (2019)
11. Matsunobu, Y., Dong, S., Lee, H.: Myrocks: LSM-tree database storage engine serving facebook's social graph. Proc. VLDB Endowment **13**(12), 3217–3230 (2020)
12. Nebula graph (2021). https://github.com/vesoft-inc/nebula-graph
13. Taft, R., et al.: Cockroachdb: the resilient geo-distributed sql database. In: Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data, pp. 1493–1509 (2020)

14. Van Aken, D., Pavlo, A., Gordon, G.J., Zhang, B.: Automatic database management system tuning through large-scale machine learning. In: Proceedings of the 2017 ACM International Conference on Management of Data, pp. 1009–1024 (2017)
15. YCSB-C (2018). https://github.com/basicthinker/YCSB-C
16. Zhang, J., et al.: An end-to-end automatic cloud database tuning system using deep reinforcement learning. In: Proceedings of the 2019 International Conference on Management of Data, pp. 415–432 (2019)
17. Zhu, Y., et al.: Bestconfig: tapping the performance potential of systems via automatic configuration tuning. In: Proceedings of the 2017 Symposium on Cloud Computing, pp. 338–350 (2017)