RTune: A RocksDB Tuning System with Deep Genetic Algorithm

HUIJUN JIN
Computer Science
Yonsei University
Seoul, Korea
jinhuijun@yonsei.ac.kr

Jieun Lee
Computer Science
Yonsei University
Seoul, Korea
jieun199624@yonsei.ac.kr

Sanghyun Park†
Computer Science
Yonsei University
Seoul, Korea
sanghyun@yonsei.ac.kr

ABSTRACT

Database systems typically have many knobs that must be configured by database administrators to achieve high performance. RocksDB achieves fast data writing performance using a log-structured merge-tree. This database contains many knobs related to write and space amplification, which are important performance indicators in RocksDB. Previously, it was proved that significant performance improvements could be achieved by tuning database knobs. However, tuning multiple knobs simultaneously is a laborious task owing to the large number of potential configuration combinations and trade-offs.

To address this problem, we built a tuning system for RocksDB. First, we generated a valuable RocksDB data repository for analysis and tuning. To find the workload that is most similar to a target workload, we created a new representation for workloads. We then applied the Mahalanobis distance to create a combined workload that is as close to the original target workload as possible. Subsequently, we trained a deep neural network model with the combined workload and used it as the fitness function of a genetic algorithm. Finally, we applied the genetic algorithm to find the best solution for the original target workload. The experimental results demonstrated that the proposed system achieved a significant performance improvement for various target workloads.

CCS CONCEPTS

• Computing methodologies → Machine learning → Machine learning approaches → Neural networks; • Information systems → Data management systems → Database administration → Database performance evaluation;

†Author Footnote to be captured as the corresponding author

GECCO'22, July 9-13, 2022, Boston, MA, USA
© 2022 Association for Computing Machinery.
ACM ISBN 978-1-4503-9237-2/22/07...$15.00
https://doi.org/10.1145/3512290.3528726

KEYWORDS

Database Optimization, Log-Structured Merge-Tree, Genetic Algorithm, Write Amplification, Space Amplification

ACM Reference format:

HUIJUN JIN, Jieun Lee and Sanghyun Park. 2022. RTune: A RocksDB Tuning System with Deep Genetic Algorithm. In *Proceedings of the Genetic and Evolutionary Computation Conference 2022 (GECCO '22). ACM, Boston, MA, USA, 9 pages.* https://doi.org/10.1145/3512290.3528726

1 Introduction

In the era of big data, unstructured data are being produced and consumed in large quantities. The use of traditional relational databases to process these unstructured data is difficult. To solve this problem, key-value databases have been proposed as a means of storing and managing unstructured data in the form of key-value pairs.

RocksDB is a key-value database that uses a log-structured merge-tree (LSM-tree) for fast data processing [1, 2]. The LSM-tree achieves excellent write performance with sequential writing, while it requires additional write operations and space usage for data compaction [3, 4, 5, 6]. Additional write operations reduce the data processing performance of a database and the lifespan of a storage device. Furthermore, additional writing results in the storage of more data than expected, leading to increased space usage [7, 8, 9].

Database tuning has been studied extensively. The difficulty in database tuning is that too many factors influence the final performance of a database, such as knobs (e.g., write_buffer_size), workload information (e.g., key size: 16 B, value size: 1K, read-write ratio: 1:9), and hardware settings (e.g., memory size) [10]. Database tuning is thus challenging, even for database experts. RocksDB also provides hundreds of knobs that can be tuned. To address this problem, we developed a system with a deep genetic algorithm (GA) for RocksDB knob optimization. We generated a valuable RocksDB data repository with 16 different workloads simulating real-world situations and used these workloads for further study. In our study, we used a novel method to represent a workload and created combined workloads with Mahalanobis distance that are as close to the target workload as possible and assumed that the combined workload is the original target workload. Then, we used our original RocksDB data repository to train a deep neural network (DNN) model to predict the external metrics from a configuration. Finally, we applied this DNN model as the fitness function of a GA to find the best solution. Following the system outlined above, we successfully determined the best solutions and proved that these solutions are generally the best through many comparative experiments.

The main contributions of this study can be summarized as follows:

• ***Data Repository Generation***: We generated a valuable RocksDB data repository including 20,000 random configurations for 16 workloads over three months.

• ***New Representation of a Workload***: We used five statistics (average, variance, 1st quartile, 2nd quartile, and 3rd quartile) to simplify the workload representation and reduce the dimensions of the search space.

• ***Created Combined workloads***: We applied the Mahalanobis distance to the data repository and the target workloads as well as created workloads that were most similar to the target workloads.

• ***Novel Score Function***: We proposed a novel score function for a DNN model to solve the multi-optimization problem. The score function was expressed as the sum of the proportion of the predicted values and the default values of the four external metrics, and it was sufficient to represent the overall performance of RocksDB.

• ***Short Tuning Time with Good Performance***: The proposed system can successfully determine the best configuration within a short tuning time. The final performance achieved by the system was better than that with the default settings, Facebook recommendations, and database administrator (DBA) recommendations.

The remainder of this paper is organized as follows. Section 2 presents the background of the LSM-tree, write amplification, space amplification, and Mahalanobis distance. Section 3 describes our model, along with its motivation, methods, and design. Section 4 presents an evaluation of the experimental setup and the results of comparative experiments. Section 5 presents the work related to our model. Finally, Section 6 presents the conclusions and future work.

2 Background

2.1 LSM-Tree and RocksDB

RocksDB is a disk-based database that uses the LSM-tree as its basic structure. The LSM-tree is a tree-shaped structure that contains several levels. Compaction is an operation that an LSM-tree uses to manage the data and maintain the structure of the

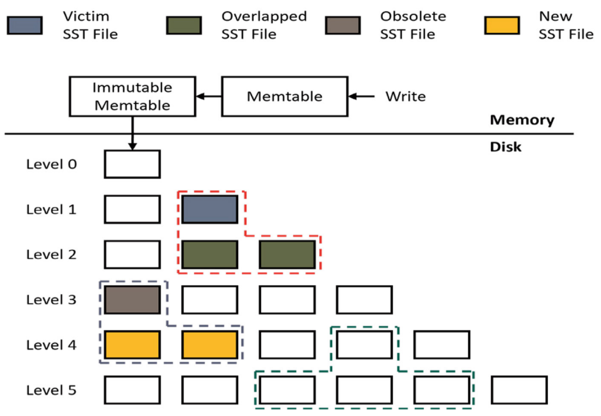

Figure 1: LSM-Tree and compaction. The dashed box represents three compaction processes in different phases.

LSM-tree, and the smallest file unit of RocksDB is known as a static sorted table (SST).

The compaction process is illustrated in Figure 1. Three compaction processes are depicted in Figure 1 (red, blue, and green dashed-line boxes). The red dashed-line box represents the beginning status of one compaction, and the blue dashed-line box represents the end status of another compaction.

The compaction process is triggered when a level exceeds its maximum capacity. A victim SST file that exists at level 1 is selected when compaction is triggered. The overlapped SST files that have overlapping key ranges with the victim SST file are additionally selected at level 2.

After compaction, new SST files are created by victim SST files in level 3, and the overlapped SST files in level 4 through merge sort and are sorted at level 4. Obsolete SST files are generated during compaction and are later deleted through the garbage collection system.

Compactions are executed simultaneously at many levels because RocksDB is a multi-threaded database, which leads to a problem that is discussed in the next section.

2.2 Write and Space Amplification

Two amplifications occur during the process of compaction: write amplification (WA) and space amplification (SA).

To send a victim SST file at level n to a lower level, RocksDB must locate the overlapped SST files at level n + 1 and apply merge sort to both the victim and overlapped SST files. Additional write operations are performed during this process, and this phenomenon is known as WA.

After a compaction process, invalid SST files that are no longer needed for users occupy the storage space until they are deleted via garbage collection. In addition, the same data cannot exist at the same level because of the principle of the LSM-tree, while they can exist at different levels. This results in the same data being duplicated at several different levels. These two phenomena of unnecessary space occupancy are known as SA.

RocksDB is a multi-threaded database, which implies that RocksDB performs many compaction operations simultaneously, and this leads to an exponential increase in the amount of write and space amplifications. WA decreases data processing performance and shortens the lifespan of storage devices because of excessive disk input/output operations. SA increases the burden on storage devices in terms of database operations. Avoiding unnecessary disk input/output operations on storage devices is important because storage devices usually have limitations in terms of the number of write operations. In addition, reducing unnecessary space occupancy can increase the cost-effectiveness of storage devices.

2.3 Mahalanobis Distance

Distance is a measure of how far two points are from each other. In this paper, we used distance to calculate the similarity between workloads for combined workload creation. A typical example is a Euclidean distance, which is the distance between two points on the coordinate plane according to the Pythagorean theorem. Euclidean distance is simple and efficient for finding the distance between two points; however, it has a disadvantage in determining the distance between data.

The Mahalanobis distance is a value indicating how many times the standard deviation is the distance from the mean value [11]. It is a method to quantify how rare and strange a value is, and it is mainly used to distinguish whether data are fake or real. Thus, this distance is calculated using the covariance matrix for the Euclidean distance. The formula is as follows:

$$D_M(\vec{x}) = \sqrt{(\vec{x} - \vec{\mu})^{\mathrm{T}} \mathbf{S}^{-1} (\vec{x} - \vec{\mu})} \qquad (1)$$

3 Methodology

Figure 2 shows an overview of the proposed system. As depicted in Figure 2, in the data generation phase, we first executed 20,000 randomly generated configurations in 16 different workloads. It comprises 16 different workloads, each with 20,000 pairs of configurations, internal metrics (IM, e.g., rocksdb.block.cache.miss), and external metrics (EM, e.g., TIME).

Second, in the workload representation phase, we applied five statistics (average, variance, 1st quartile, 2nd quartile, and 3rd quartile) to the internal metrics of each workload to create new workload representations.

Third, with the new representation of 16 workloads and the target workload, we applied the Mahalanobis distance to calculate the distance from the target workload to each of the 16 workloads and regarded the distance as the similarity.

Fourth, we used the data repository generated in the first step and the distances created in the third step to generate a combined workload that is as close to the target workload as possible and assumed that the combined workload is the target workload.

Fifth, we trained the DNN model with the combined workload of its 20,000 "configuration – internal metrics – external metrics" pairs.

Finally, we applied the trained DNN model as the fitness function of the GA, which was used to recommend the best

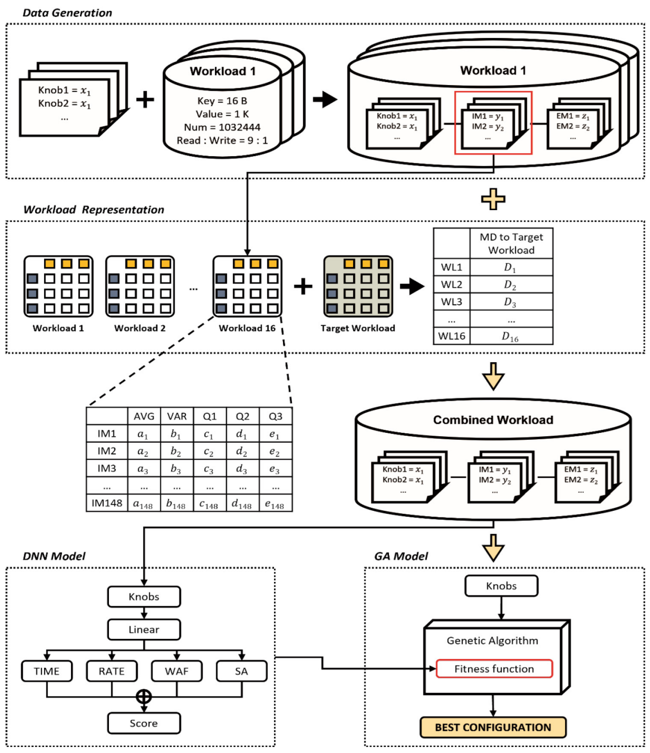

Figure 2: Overview of proposed model architecture

configuration; this configuration can achieve excellent performance with the original target workload.

3.1 Data Generation

We used the DB_Bench benchmarking tool for RocksDB data repository generation. RocksDB provides DB_Bench for testing and analyzing its performance [12]. DB_Bench is provided by RocksDB along with all the knobs, such as level size, compression techniques, data size, and memory size. DB_Bench can be executed with the "--statistics" option to obtain internal and external metrics. Internal metrics indicate information such as configuration values, workload, and I/O status, and external metrics indicate the level of the benchmark performance. We used the total execution time (TIME), data processing rate (RATE), write amplification factor (WAF), and SA as external metrics for this study, and the details of these metrics will be discussed in subsequent sections.

RocksDB has more than 200 tunable knobs. Facebook noted that there are more than 20 important knobs for performance improvement [13]. We selected 22 knobs with the help of RocksDB experts. To accelerate data repository generation and generate more data, we shrank the RocksDB LSM-tree size to 1/64 compared to the default size. We set the range of each knob according to the size of the shrunk LSM-tree and generated 20,000 random configurations based on them. The total data size for each workload was set to 1 GB, and they consisted of different value sizes, numbers of entries, read-write ratios, and update

Table 1: Basic workloads.

Workload Index	Value Size (B), # of Entry	Read : Write	Update
0	1024, 1032444	9 : 1	-
1	1024, 1032444	1 : 1	-
2	1024, 1032444	1 : 9	-
3	1024, 1032444	-	TRUE
4	4096, 261124	9 : 1	-
5	4096, 261124	1 : 1	-
6	4096, 261124	1 : 9	-
7	4096, 261124	-	TRUE
8	16384, 65472	9 : 1	-
9	16384, 65472	1 : 1	-
10	16384, 65472	1 : 9	-
11	16384, 65472	-	TRUE
12	65536, 16380	9 : 1	-
13	65536, 16380	1 : 1	-
14	65536, 16380	1 : 9	-
15	65536, 16380	-	TRUE

Table 2: Original workload representation.

Configuration #	1	2	…	20000
IM 1	12965	13040	…	14586
IM 2	1239	837	…	297
…	…	…	…	…
IM n	44	63	78	208

Table 3: New workload representation.

	Average	Variance	1st Quartile	2nd Quartile	3rd Quartile
IM 1	13544	55615	10513	13448	15798
IM 2	834	2315	564	912	1132
…	…	…	…	…	…
IM n	80	1213	68	81	121

option. The key size was fixed at 16 B and the value sizes were set to 1, 4, 16, and 64 KB. The read-write ratios were [read 90%, write 10%], [read 50%, write 50%], and [read 10%, write 90%]. These hyperparameters can be set using the options provided by DB_Bench. Table 1 presents the details of each workload. With 20,000 random configurations and 16 different workloads, we generated a total of 320,000 [Configuration – Internal Metrics – External Metrics] pairs of RocksDB data.

3.2 Design

3.2.1 Workload Representation. We have 20,000 [Configuration – Internal Metrics – External Metrics] pairs for each workload, which is obtained from the data repository generation of RocksDB. Internal metrics include information for a workload, such as the key size, total entry number, and read-write ratio. Thus, a workload can be represented by internal metrics with 20,000 configurations. Table 2 shows an example of a workload represented by internal metrics. The size of the table is large, and it is expensive to proceed with various calculations [14, 15].

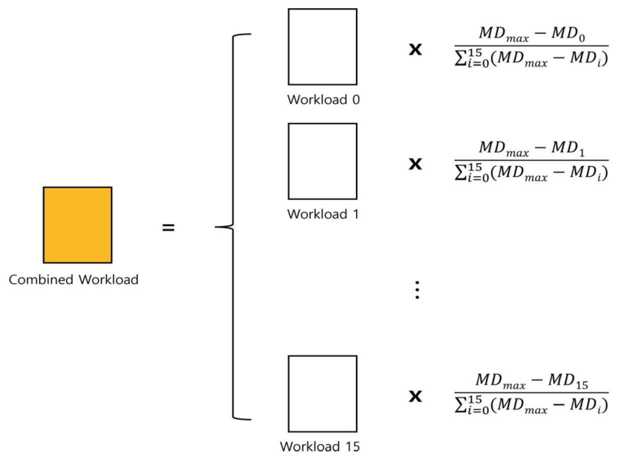
Combined Workload
=
Workload 0
×
$\frac{MD_{max} - MD_0}{\sum_{i=0}^{15}(MD_{max} - MD_i)}$
Workload 1
×
$\frac{MD_{max} - MD_1}{\sum_{i=0}^{15}(MD_{max} - MD_i)}$
⋮
Workload 15
×
$\frac{MD_{max} - MD_{15}}{\sum_{i=0}^{15}(MD_{max} - MD_i)}$

Figure 3: Combined workload calculation process. MD corresponds to Mahalanobis distance.

To solve this problem, we calculated five statistics of 20,000 data points for each internal metric. The five statistics are average, variance, 1st quartile, 2nd quartile, and 3rd quartile. Table 3 presents a new representation of a workload. This is because the values of the internal metrics of a particular workload should not vary significantly because of configuration changes.

To obtain the workload representation of a target workload, we require at least 20 of the target workload data points. The method for generating 20 data points of a target workload is similar to the RocksDB data repository generation mentioned previously. To accelerate the data repository generation speed of a target workload, we selected the 20 fastest configurations among the 320,000 RocksDB data repository generation results to generate data.

There are several advantages to cover the data loss of dimension reduction. First, it is cheap to apply various calculations a with new workload representation. Second, it is easy to get 5 statistics of a target workload with a small sample(20). Third, we were able to find the optimal configuration with the new workload representation.

3.2.2 Combined Workload. We applied our new workload representation technique to basic and target workloads. Subsequently, we applied the Mahalanobis distance to these workloads to calculate the distance from the target workload to each basic workload and regarded the distance as the similarity. For example, to obtain the distance between basic workload 1 and the target workload, we calculated the Mahalanobis distance of the first internal metric for workload 1 and the target workload. We applied this method to all 148 internal metrics of both workload 1 and the target workload and then added these distances to obtain the distance between workload 1 and the target workload. Following the above method, we obtained a matrix of distances between the basic workloads and the target workload. We regarded the matrix as the similarity matrix from the target

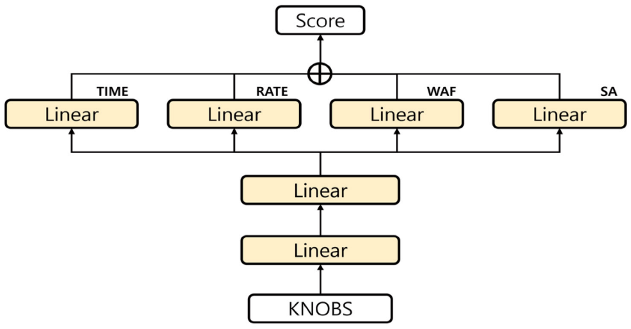

Figure 4: DNN model structure. The score is calculated according to Equation (2).

workload to each basic workload. The combined workload has a total number of 20,000 [Configuration – Internal Metrics – External Metrics] pairs, similar to other workloads. As depicted in Figure 3, the amount of data for each basic workload to be included in the combined workload is determined using this matrix. We calculated the difference between the maximum distance and each distance for scaling purposes. Therefore, the combined workload consists of parts of each basic workload based on the similarity matrix. By using the combined workload, we were able to include the knowledge of the target workload into the combined workload which is used for training the DNN model.

3.2.3 Deep Neural Network Model. Subsequently, we considered the combined workload as the target workload and used the combined workload to train the DNN model. The DNN model takes the configurations as input and predicts four external metrics (TIME, RATE, WAF, and SA). As depicted in Figure 4, we used two fully connected layers and four linear layers for each external metric. The score is calculated as the sum of the proportions of the Default (D) and predicted (P) values of each metric. Equation (2) represents this, and the weight of each term is set to 0.25. The external metrics were scaled appropriately to avoid the problem of large deviations.

$$Score = \alpha_1 \frac{TIME_D}{TIME_P} + \alpha_2 \frac{RATE_P}{RATE_D} + \alpha_3 \frac{WAF_D}{WAF_P} + \alpha_4 \frac{SA_D}{SA_P} \quad (2)$$

3.2.4 Genetic Algorithm. A GA is a model that imitates the evolution of creatures and is a technique used for optimization problems. GAs use terms such as gene, chromosome, mutation, generation, population, and crossover [16, 17, 18].

Figure 5 depicts a flowchart of the GA used in this study. The trained DNN model described in the previous section was used as the fitness function of the GA. Four different chromosomes (A, B, C, and D) were used as the initial population. Each chromosome is a configuration, and each gene is an individual knob. The initial population is then evaluated through the DNN model, and it outputs the population that is ranked by the fitness score. The GA divides the ranked population into two groups by fitness score: the preserved group (B, C) with high scores and the discarded group

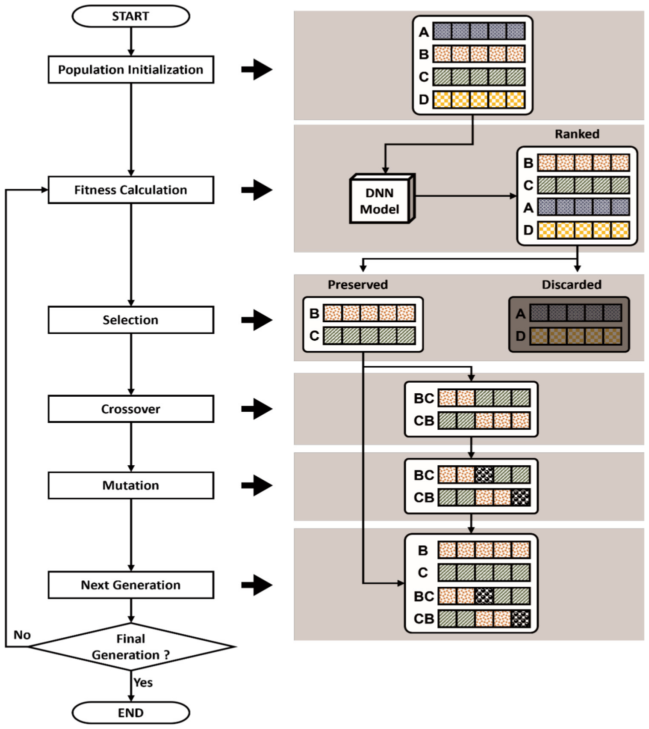

Figure 5: Genetic Algorithm for RTune. A trained DNN model with a combined workload is used as the fitness function of the Genetic Algorithm.

(A, D) with low scores. The configurations in the discarded groups are no longer used in the algorithm, whereas the preserved configurations survive in the present generation. To refill the population, the GA performs crossover and mutation procedures on the preserved configurations. The GA performs crossovers of the knobs in configurations B and C to create new configurations, namely, BC and CB. After the crossover, a knob mutation occurs with a certain probability. The mutant knob brings a new value in the range of itself, and this leads to a diversity of configurations. In addition, mutations can render the algorithm less reliant on the diversity of the initial population. Therefore, configurations BC and CD with the mutant knobs and the preserved configurations become the new population and enter the next generation. The new population is again evaluated through the DNN model, and configurations with high fitness scores continue to survive. The GA stops when it reaches the final generation, and with each generation, the configuration with the highest fitness score becomes closer to the optimal solution.

DNN model and GA model play different roles in this study. DNN predicts four external metrics from configurations while GA finds the optimal value through crossover and mutation. It is possible to use GA alone to find the optimal configuration. The problem is that it will take too much time to finish the algorithm since GA has to estimate the candidate configurations through DB_Bench every time. Even for a light workload like the one described in this paper (Size of 1GB), it usually takes several minutes or even hours to finish every benchmarking task. DNN

Table 4: Target workloads.

Workload Index	Value Size (B), # of Entry	Read : Write	Update
16	8192, 130816	7 : 3	-
17	8192, 130816	3 : 7	-
18	8192, 130816	-	True
19	32768, 32752	7 : 3	-
20	32768, 32752	3 : 7	-
21	32768, 32752	-	True

Table 5: Hyperparameters of DNN model.

Optimizer	Adamw
Learning Rate	0.0002
Epoch	300
Loss Function	MSE
X Scaler	MinMaxScaler
Y Scaler	StandardScaler
Layer	(22, 64, 16, 1)
Activation Function	ReLU

Table 6: Hyperparameters of GA model.

Mutation Ratio	0.4
Crossover Ratio	0.5
Population Size	128
Generation	1000
Selection Algorithm	Rank Selection
Selection Size	64

can solve this problem. Although it takes some time to train the model, still it is much faster than going through DB_Bench for every candidate configuration. Therefore, we use the DNN model to evaluate candidate configurations in the process of GA to accelerate the convergence of the algorithm.

4 Evaluation

4.1 Experimental Setup

We used the DB_Bench benchmarking tool for RocksDB performance measurements. The specific workloads for the experiments and experimental environments are presented in the following sections. We conducted several comparative experiments with our model including overall performance comparison, workload combining and pruned internal metrics comparison, number of knobs comparison, different weights comparison, cosine similarity and Mahalanobis distance comparison, and Bayesian optimization comparison. All of the code is accessible on our GitHub repository (https://github.com/addb-swstarlab/RocksDB-Optimization-wDNN).

4.1.1 Target Workloads Information. Sixteen workloads were generated using the methods described in the previous sections. In addition, we generated six target workloads for experimental purposes, and the target workloads had different value sizes and read-write ratios from the basic workloads. The details of the target workloads are presented in Table 4.

4.1.2 Environment Setting. The experimental environment used in this study is detailed in supplement material. The environment is used for RocksDB data repository generation, DNN model training, and GA training.

4.1.3 External Metrics. The purpose of this study was to improve the overall performance of RocksDB by selecting four performance indicators, namely, WAF, SA, RATE, and TIME, to analyze the changes in the performance of RocksDB based on various knobs. We selected RATE and TIME to meet the service acceptance criteria when improving the WAF and SA. For example, it could take considerable time to derive the value of the WAF ideally. In this case, it is inappropriate to use the model in real-life situations.

WAF is a performance indicator representing the ratio of the amount of data written to the storage (physical data size) and the amount of data written to the database (logical data size). WAF describes the amount of additional data written during the use of RocksDB. This is calculated as follows:

$$WAF = \frac{Physical\ data\ size}{Logical\ data\ size} \tag{3}$$

SA is measured from the size of the data recorded in the actual LSM-tree. Because an LSM-tree contains valid and invalid SST files simultaneously, the size of the data recorded in an LSM-tree can be used as an effective metric for the actual size of physical space exploited by RocksDB.

The RATE refers to the number of operations processed by RocksDB per second, such as compaction, compression, read, and write operations.

The TIME refers to the time interval from the start of the data recording to the end. RATE and TIME were not directly proportional in this experiment. For example, RATE increased when the size of the buffer allocated to RocksDB increased; however, TIME remained the same even when the number of operations increased, owing to the large compression ratio.

4.1.4 Hyperparameters in DNN and GA. To achieve the best performance, we conducted hundreds of experiments and determined the best hyperparameters for the DNN and GA models based on the existing knowledge. Tables 5 and 6 present the hyperparameters of the DNN and GA models [19].

4.2 Results

To avoid the effect of some large-range metrics, we applied the geometric mean to the four external metrics for performance comparison instead of the arithmetic mean. Equation (4) shows the score equation where D represents the default value while A represents the actual value of each metric. For the figures in this

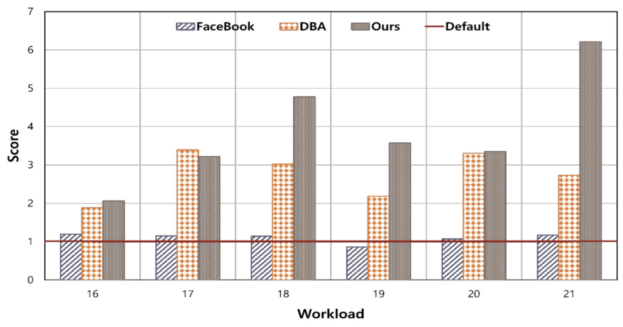

Figure 6: Overall performance comparison

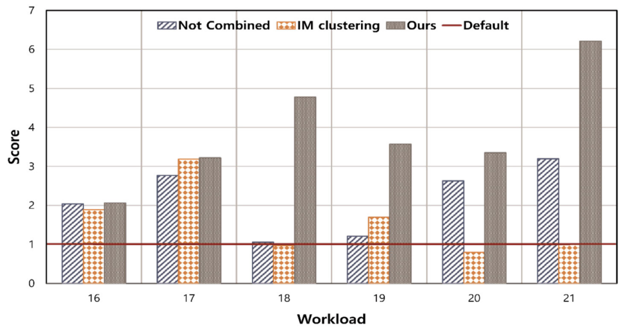

Figure 7: Workload combining and pruned internal metrics comparison

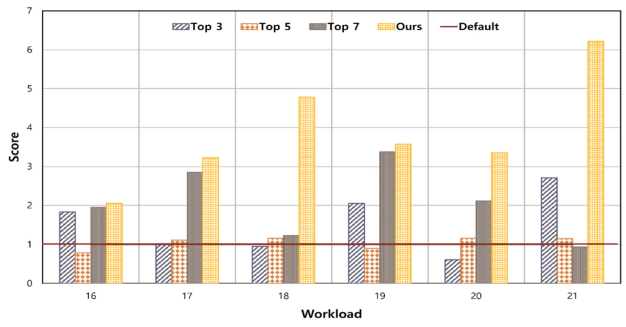

Figure 8: Number of knobs comparison

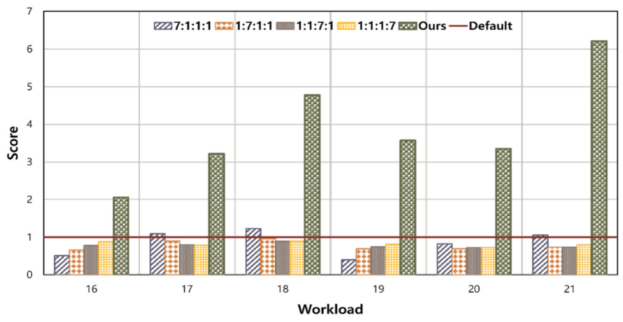

Figure 9: Different weights comparison

section, the y-axis corresponds to the score of EM while the x-axis corresponds to 6 target workloads. The performance of the default setting is indicated as a red line pointing to 1 because it is the baseline performance among all target workloads according to the performance score formula.

$$Score = \sqrt[4]{\frac{TIME_D}{TIME_A} \times \frac{RATE_A}{RATE_D} \times \frac{WAF_D}{WAF_A} \times \frac{SA_D}{SA_A}} \tag{4}$$

4.2.1 Overall Comparison. Figure 6 depicts the overall comparison of six target workloads among default settings, Facebook recommended configurations, DBA recommendations, and the proposed model. The results show that the proposed model achieved the best performance among five target workloads and was slightly lower than DBA in the 17th workload.

4.2.2 Combined Workload and Pruned Internal Metrics. In Figure 7, "Not Combined" depicts using the closest workload as the new target workload instead of creating a combined workload based on the distance to the target workload in the "workload representation" phase. A previous study used some of the internal metrics for database tuning [20]. "IM clustering" depicts using pruned internal metrics by k-means clustering with a size of 50 before the "workload representation" phase. The results illustrate that our model exhibits the best performance among all target workloads. This is because it is better to describe a workload using a combined workload and full internal metrics to represent a workload. After all, it contains more knowledge about the target workload.

4.2.3 Number of Knobs. It has been proven in a previous study that several knobs can be used to achieve good performance [21]. Figure 8 depicts the difference in performance, depending on the number of knobs used for tuning. We conducted a comparative experiment by pruning the knobs with three, five, and seven knobs. The knobs were selected in order of importance through a random forest that extracts their feature importance. Our model generally shows the best performance for all target workloads. The model with the top seven knobs has similar performance to our model in target workloads 16, 17, and 19; however, it is unstable in other target workloads.

4.2.4 Weights. Figure 9 depicts the performance comparison experiment using four different weight pairs in Equation (2). The default value of the weight is set to 0.25 equally. Our model successfully achieved the best performance for all target workloads. However, the remaining four models could barely reach the performance of the default setting. This is because it is difficult to achieve a high-performance score with only one external metric when attempting to increase the weight of the external metric. In addition, the high-weighted external metric is not much higher than the same external metric in the case of other models. This is because one particular metric cannot be extremely high because of the trade-offs between the external metrics

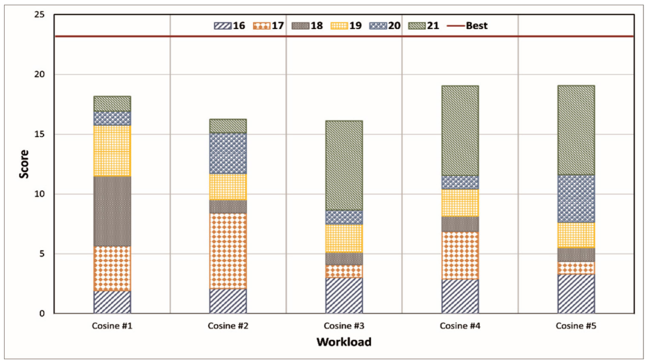

Figure 11: Cosine similarity and Mahalanobis distance comparison

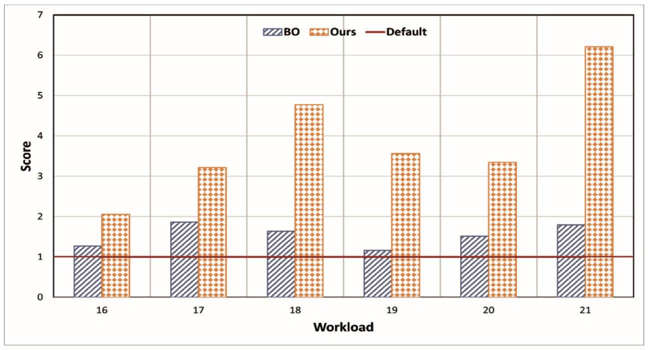

Figure 12: Bayesian optimization comparison

4.2.5 Distance Method. Figure 12 depicts the five experiments using cosine similarity instead of Mahalanobis distance in the phase of making combined workloads. The performance gained by the cosine similarity was good but cannot reach the best score which is achieved by Mahalanobis distance in all experiments. As it showed in experiment #3, cosine similarity can achieve good performance in target workload 16, 19, and 21, but had a low score in target workload 17, 18, and 20. This means using cosine similarity works sometimes, while it is not stable for all cases.

4.2.6 Bayesian Optimization. We conducted an experiment compared with Bayesian optimization(BO) instead of GA. Bayesian optimization is a well-known technique for finding the optimal solution. Sami Alabed and Eiko Yoneki suggested that using 10 RocksDB knobs for tuning can achieve good performance improvement [14]. They manually selected these 10 knobs and applied BO to find the best configuration. We used all of these knobs for this experiment. Figure 13 depicts the score comparison over 6 target workloads. The scores gained by using BO cannot overperform our model. In addition, for target workload 19, the score barely exceeds the default line. We believe the reason for such a result is that the method proposed in the paper did not include enough information and thus missed important knowledge such as knobs and external metrics.

5 Related Work

Tuning database knobs is a crucial task for improving database performance [23, 24, 25]. The goal of database knob tuning is to find the optimal knobs for a specific workload that can achieve the best performance. Machine learning and reinforcement learning techniques have been used for database auto-tuning tasks in past years [26, 27, 28, 29]. OtterTune is a tuning system that uses machine learning techniques to find the most important knobs, match previously unseen workloads to known workloads for knob recommendation [19]. It finds the main features of known workloads with factor analysis and k-means clustering and it identifies the top important knobs using a lasso algorithm. Finally, OtterTune recommends the best using the Gaussian process. OtterTune relied heavily on a good data repository in previous experience to match the unseen workload correctly; however, the data repository is not open-source and hard to obtain by users. To address the large search space problem, BestConfig uses the divide-and-diverge sampling method and the recursive bound-and-search algorithm to divide knob spaces into several subspaces [30]. CDBTune applies the reinforcement learning method to find the best configurations with little samples [31]. CDBTune uses a reward feedback method to accelerate the speed of convergence and improve the accuracy of prediction. Offline training and online tuning are two main steps of CDBTune. In the offline tuning phase, it trains the model with a standard benchmark and workloads. Then CDBTune uses the pre-trained model to tune a specific database in the online tuning phase. The total tuning process takes several hours. Our model, RTune, created a new representation of a workload combined with Mahalanobis distance to successfully captured the features of a workload and put them into a DNN model to train it. Then we used a GA to find the best configuration with the trained DNN model to accelerate the process. RTune successfully found the best solutions in 6 target workloads.

6 Conclusion

In this study, we generated a valuable RocksDB data repository for data analysis and tuning. Subsequently, we applied the Mahalanobis distance to find the combined workload that was most similar to the target workload. We trained a DNN model with RocksDB data repository (knob values paired with internal and external metrics). The DNN model was used as the fitness function of the GA. Using the above system, we successfully determined the best solutions for different workloads.

ACKNOWLEDGMENTS

This work was supported by Institute of Information & communications Technology Planning & Evaluation(IITP) grant funded by the Korea government(MSIT) (IITP-2017-0-00477, (SW starlab) Research and development of the high performance in-memory distributed DBMS based on flash memory storage in IoT environment)

REFERENCES

[1] F. Mei, Q. Cao, H. Jiang, and L. Tian, "LSM-tree managed storage for large-scale key-value store," *IEEE Transactions on Parallel and Distributed Systems,* vol. 30, no. 2, pp. 400-414, 2008.

[2] RocksDB: A persistent key-value store for fast storage environments. Available: https://rocksdb.org.

[3] P. O'Neil, E. Cheng, D. Gawlick, and E. O'Neil, "The log-structured merge-tree (LSM-tree)," *Acta Informatica,* vol. 33, no. 4, pp. 351-385, 1996.

[4] K. Ouaknine, O. Agra, and Z. Guz, "Optimization of rocksdb for redis on flash," *Proceedings of the International Conference on Compute and Data Analysis,* pp. 155-161, 2017.

[5] Z. Cao, S. Dong, S. Vemuri, and D. H. Du, "Characterizing, modeling, and benchmarking rocksdb key-value workloads at facebook," *18th {USENIX} Conference on File and Storage Technologies ({FAST} 20),* pp. 209-223. 2020.

[6] H. Kim, J. H. Park, S. H. Jung, and S. W. Lee, "Optimizing RocksDB for Better Read Throughput in Blockchain Systems," *23rd International Computer Science and Engineering Conference (ICSEC),* pp. 305-309, 2019.

[7] X. Y. Hu, E. Eleftheriou, R. Haas, I. Iliadis, and R. Pletka, "Write amplification analysis in flash-based solid state drives," *Proceedings of SYSTOR 2009: The Israeli Experimental Systems Conference,* pp. 1-9, 2009.

[8] Y. Lu, J. Shu, W. Zheng, "Extending the lifetime of flash-based storage through reducing write amplification from file systems," *11th {USENIX} Conference on File and Storage Technologies ({FAST} 13),* 2013.

[9] S. Odeh, Y. Cassuto, "NAND flash architectures reducing write amplification through multi-write codes," *2014 30th Symposium on Mass Storage Systems and Technologies (MSST),* 2014.

[10] T. Schmied, D. Didona, A. Döring, T. Parnell, and N. Ioannou, "Towards a General Framework for ML-based Self-tuning Databases," *Proceedings of the 1st Workshop on Machine Learning and Systems*, pp. 24-30, 2021.

[11] G. J. McLachlan, "Mahalanobis distance." *Resonance* 4.6, pp. 20-26, 1999.

[12] "Benchmarking tools." Available: https://github.com/facebook/rocksdb/wiki/Benchmarking-tools.

[13] RocksDB tuning guide Available: https://github.com/EighteenZi/rocksdb_wiki/blob/master/RocksDB-Tuning-Guide.md

[14] S. Alabed, and E. Yoneki, "High-Dimensional Bayesian Optimization with Multi-Task Learning for RocksDB," *Proceedings of the 1st Workshop on Machine Learning and Systems*, pp. 111-119, 2021.

[15] Z. Yan, J. Lu, N. Chainani, and C. Lin, "Workload-Aware Performance Tuning for Autonomous DBMSs," *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. IEEE, pp. 2365-2368, 2021.

[16] D. Whitley, "A genetic algorithm tutorial," *Statistics and Computing,* vol. 4, no. 2, pp. 65-85, 1994.

[17] S. J. Mardle, S. Pascoe, and M. Tamiz, "An investigation of genetic algorithms for the optimization of multi-objective fisheries bioeconomic models," *International Transactions in Operational Research*, vol. 7, no. 1, pp. 33-49, 2000.

[18] S. Jena, P. Patro, and S. S. Behera, "Multi-Objective Optimization of Design Parameters of a Shell &Tube type Heat Exchanger using Genetic Algorithm," *International Journal of current Engineering and Technology*, vol. 3, no. 4, pp. 1379-1386, 2013.

[19] Y. Jia, and F. Chen, "Kill Two Birds with One Stone: Auto-tuning RocksDB for High Bandwidth and Low Latency," *2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS)*, IEEE, pp. 652-664, 2020.

[20] D. V. Aken, A. Pavlo, G. J. Gordon, and B. Zhang, "Automatic Database Management System Tuning Through Large-scale Machine Learning," *Proceedings of the 2017 ACM International Conference on Management of Data,* pp. 1009-1024, 2017.

[21] K. Kanellis, R. Alagappan, and S. Venkataraman, "Too many knobs to tune? towards faster database tuning by pre-selecting important knobs," *12th {USENIX} Workshop on Hot Topics in Storage and File Systems (HotStorage 20),* 2020.

[22] L. Breiman, "Random forests," *Machine Learning,* vol. 45, no. 1, pp. 5-32, 2001.

[23] Y. Ishihara, and M. Shiba, "Dynamic Configuration Tuning of Working Database Management Systems," *2020 IEEE 2nd Global Conference on Life Sciences and Technologies (LifeTech)*. IEEE, pp. 393-397, 2020.

[24] C. Shan, N. Mamoulis, R. Cheng, G. Li, X. Li, and Y. Qian, "An end-to-end deep RL framework for task arrangement in crowdsourcing platforms," *2020 IEEE 36th International Conference on Data Engineering (ICDE)*, IEEE, pp. 49-60, 2020.

[25] S. Chaudhuri, and V. Narasayya, "Self-tuning database systems: a decade of progress," *Proceedings of the 33rd international conference on Very large data bases*, pp. 3-14, 2007.

[26] G. Li, X. Zhou, S. Li, and B. Gao, "Qtune: A query-aware database tuning system with deep reinforcement learning," *Proceedings of the VLDB Endowment* 12.12, pp. 2118-2130, 2019.

[27] L. Ferreira, F. Coelho, and J. Pereira, "Self-tunable DBMS Replication with Reinforcement Learning," *IFIP International Conference on Distributed Applications and Interoperable Systems*. Springer, Cham, pp. 131-147, 2020.

[28] D. Van Aken, D. Yang, S. Brillard, A. Fiorino, B. Zhang, C. Bilien, and A. Pavlo, "An inquiry into machine learning-based automatic configuration tuning services on real-world database management systems," *Proceedings of the VLDB Endowment* 14.7, pp. 1241-1253, 2021.

[29] Z. Lin, L. Kai, Z. Cheng, and J. Wan, "RangeKV: An Efficient Key-Value Store Based on Hybrid DRAM-NVM-SSD Storage Structure," *IEEE Access* 8, pp. 154518-154529, 2020.

[30] Y. Zhu, J. Liu, M. Guo, Y. Bao, W. Ma, Z. Liu, …, and Y. Yang, "Bestconfig: tapping the performance potential of systems via automatic configuration tuning," *Proceedings of the 2017 Symposium on Cloud Computing,* pp. 338-350, 2017.

[31] J. Zhang, Y. Liu, K. Zhou, G. Li, Z. Xiao, B. Cheng, …, and Z. Li, "An end-to-end automatic cloud database tuning system using deep reinforcement learning," *Proceedings of the 2019 International Conference on Management of Data,* pp. 415-432, 2019.