J Inf Process Syst, Vol.18, No.3, pp.374~388, June 2022
https://doi.org/10.3745/JIPS.04.0244

ISSN 1976-913X (Print)
ISSN 2092-805X (Electronic)

JOURNAL OF INFORMATION PROCESSING SYSTEMS JIPS

Improvement of RocksDB Performance via Large-Scale Parameter Analysis and Optimization

Huijun Jin[1], Won Gi Choi[2], Jonghwan Choi[1], Hanseung Sung[3], and Sanghyun Park[1,*]

Abstract

Database systems usually have many parameters that must be configured by database administrators and users. RocksDB achieves fast data writing performance using a log-structured merged tree. This database has many parameters associated with write and space amplifications. Write amplification degrades the database performance, and space amplification leads to an increased storage space owing to the storage of unwanted data. Previously, it was proven that significant performance improvements can be achieved by tuning the database parameters. However, tuning the multiple parameters of a database is a laborious task owing to the large number of potential configuration combinations. To address this problem, we selected the important parameters that affect the performance of RocksDB using random forest. We then analyzed the effects of the selected parameters on write and space amplifications using analysis of variance. We used a genetic algorithm to obtain optimized values of the major parameters. The experimental results indicate an insignificant reduction (-5.64%) in the execution time when using these optimized values; however, write amplification, space amplification, and data processing rates improved considerably by 20.65%, 54.50%, and 89.68%, respectively, as compared to the performance when using the default settings.

Keywords

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

1. Introduction

Traditionally, relational databases have been used to process structured data. However, in the present era of big data, unstructured data, which are difficult to process using relational databases, are being mass-produced and consumed in large quantities. Therefore, key-value databases have been proposed to store and manage unstructured data in the form of key-value pairs to overcome the limitations of traditional database systems.

RocksDB is a disk-based key-value database that uses a log-structured merge tree (LSM-Tree) for rapid data processing [1,2]. The LSM-Tree exhibits excellent write performance by inducing sequential writing; however, it requires additional writing and space usage for data compaction [3-6]. Additional writes cause a deterioration of the processing performance of a database; moreover, excessive write operations cause a decrease in the lifespan of a solid-state drive (SSD) on which data are stored [7-9].

※ This is an Open Access article distributed under the terms of the Creative Commons Attribution Non-Commercial License (http://creativecommons.org/licenses/by-nc/3.0/) which permits unrestricted non-commercial use, distribution, and reproduction in any medium, provided the original work is properly cited.

Manuscript received December 30, 2020; first revision March 15, 2021; accepted April 26, 2021.

*** Corresponding Author:** Sanghyun Park (sanghyun@yonsei.ac.kr)
[1] Dept. of Computer Science, Yonsei University, Seoul, Korea ({jinhuijun, mathcombio, sanghyun}@yonsei.ac.kr)
[2] Korea Electronics Technology Institute (KETI), Seoul, Korea (cwk1412@keti.re.kr)
[3] Tmax Tibero R&D center, Seoul, Korea (hanseung_sung@tmax.co.kr)

Copyright© 2022 KIPS

These writes result in storage of more data than expected, which increases the burden on the storage device in terms of database operations.

Optimizing the database parameters is necessary to minimize the write amplification (WA) and space amplification (SA) problems that occur in RocksDB. However, tuning a considerable number of parameters in a database to improve its performance is a difficult task and requires considerable time and effort, even for database experts. To deal with numerous RocksDB parameters, we exploit a machine learning-based feature selection technique to identify important RocksDB parameters and utilize a genetic algorithm to explore a large search space and find optimal values efficiently.

This paper makes the following key contributions:

- The proposed method identifies important parameters that significantly affect the performance of WA and SA in RocksDB to reduce the tuning load.
- The proposed method utilizes a genetic algorithm with a specific fitness function for RocksDB to obtain the optimal values of the identified database parameters.
- The use of the optimal values determined through the proposed method reduce WA and SA and accelerate the rate of data processing, while the total data processing time remains almost the same as the time required with the default settings in RocksDB.

The remainder of this paper is organized as follows. Section 2 presents the background of RocksDB and its architecture, data storage mechanism, WA, and SA. Section 3 describes the tuning pipeline and the methods used in this study. Section 4 presents the experimental results. Section 5 presents recent related work. Finally, we present our conclusions and provide directions for future work in Section 6.

2. Background

2.1 RocksDB: Architecture

RocksDB uses the LSM-Tree as its basic structure [3]. The LSM-Tree consists of several levels composed of files, sorted by keys. It uses a compaction operation to manage the data (Fig. 1). Compaction is a writing method that enables the maintenance of the LSM-Tree structure. It is triggered when the threshold of each level is exceeded. RocksDB uses a static sorted table (SST) to store and manage persistent data. As depicted in Fig. 1(a), a victim SST file, existing at level n, is selected when compaction is triggered. SST files, of which the key ranges overlap with those of the victim SST file, are additionally selected at level $n + 1$. As depicted in Fig. 1(b), the selected SST files are created as newly generated SST files through merge sort and stored at level $n + 1$. The SST files selected for compaction were managed as invalid SST files and later deleted through the garbage collection system.

2.2 Write and Space Amplifications

Compaction is the task of managing space in the LSM-Tree. The process is performed by sending files, existing at a level that has reached the threshold, to a lower level. As depicted in Fig. 1, 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 called the WA [7]. Because RocksDB is a multi-threaded

database, it performs many compaction operations simultaneously, which leads to an exponential increase in the number of write operations. The data processing performance deteriorates because excessive disk input/output operations are performed. In addition, WA degrades the data processing performance and shortens the lifespan of flash memory-based devices such as SSDs. Because SSDs have limitations on the number of writes for writing files onto them, it is crucial to avoid unnecessary disk input/output operations.

In addition to the WA phenomenon, RocksDB has another problem involving data storage space utilization. As depicted in Fig. 1, invalid SST files that are no longer needed occupy the disk space until they are deleted by garbage collection. This space occupancy phenomenon is called SA. These accumulated invalid SST files occupy a considerable amount of space because numerous compaction operations are performed simultaneous.

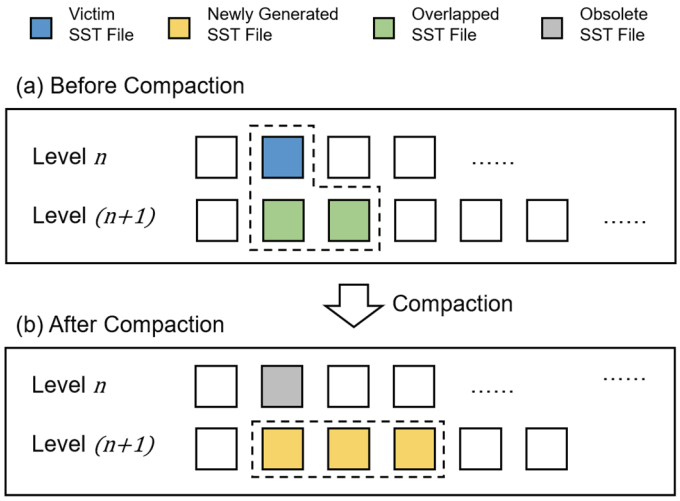

Fig. 1. RocksDB compaction diagram. (a) LSM-Tree state of RocksDB before compaction. (b) LSM-Tree state of RocksDB after compaction. Dash-line box represents SST files involved in compaction procedure.

3. Method

3.1 Tuning Pipeline

Fig. 2 shows the experimental procedure used in this study. First, we generated samples with 2,000 random parameter combinations using db_bench (see details in Section 4). Second, we calculated the contribution of each parameter using a random forest and selected the influential parameters with the greatest contribution. Third, we confirmed the correlation between the parameters and the performance of RocksDB using analysis of variance (ANOVA). Finally, we used a genetic algorithm to find the optimal values of the selected parameters and achieved a significant improvement in the performance of RocksDB compared with the performance using default settings. The specific method is described in the following sections.

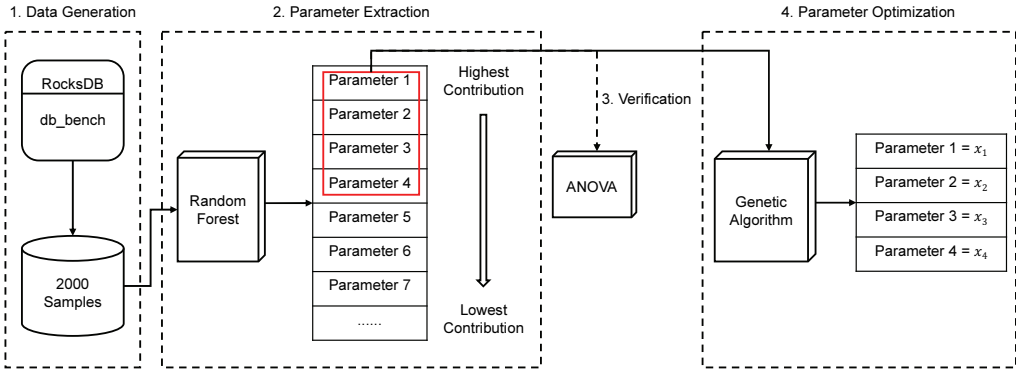

Fig. 2. Tuning pipeline. Dash-line boxes represent procedure steps.

3.2 Random Forest and Contribution Analysis

As depicted in Fig. 3, the random forest generates multiple training samples through bootstrapping and performs prediction or regression analysis by combining decision trees trained with each training sample. It is a machine-learning model [10]. The random forest classifier calculates the impurity of each feature using entropy or the Gini index and builds a feature node that reduces the impurity to the maximum extent. Similarly, the random forest regressor uses the mean squared error to define the impurity and builds the optimal branch.

The random forest can calculate the mean decrease impurity (MDI) using the impurity of each feature, which is calculated from several decision trees; based on this, the feature contribution to classification and regression can be calculated [5]. A large average impurity reduction implies that the feature is important. The feature contribution should be examined through additional analysis.

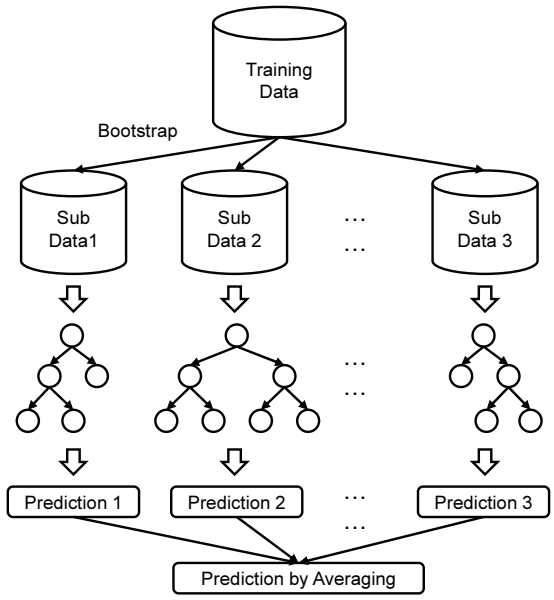

Fig. 3. Diagram of random forest.

3.3 Regression Model Coefficient of Determination

We calculated the coefficient of determination (R^2) to evaluate whether the random forest regression model learned the training samples efficiently. R^2 is calculated as the ratio of the total sum of squares of the actual values (y_i) and the sum of squares of residuals to the predicted value (f_i), as shown in Eq. (1), where $\bar{y}$ is the average of the actual values. The value of R^2 is between 0 and 1; if the value is closer to 1, the model explains the training data more effectively.

$$R^2 = 1 - \frac{\sum_i (y_i - f_i)^2}{\sum_i (y_i - \bar{y})^2} \tag{1}$$

3.4 Analysis of Variance

We used ANOVA to analyze the differences in each of the performance indicators, based on categorical variables. ANOVA statistically tests the difference in the distribution of dependent variables between the items by calculating the ratio of the sum of the variances of the dependent variable based on the category items to the sum of the variances of the total values of the dependent variable for a given categorical variable [11]. The significance probability of the p-value was calculated using test statistics. Generally, when the p-value is less than 0.05, it is interpreted as a difference in performance based on the variable value.

3.5 Genetic Algorithm

The genetic algorithm is a model that imitates the evolution of living beings, and terms such as mutation, crossover, generation, chromosome, and gene are used. Genetic algorithm is a technique used to solve optimization problems [12-14].

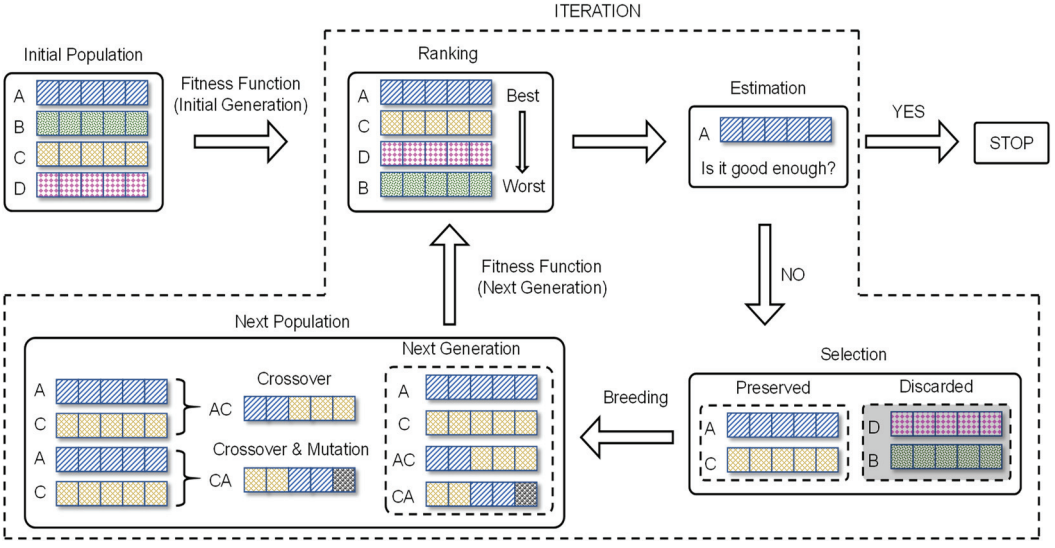

Fig. 4. Genetic algorithm. Dash-line box represents iteration.

Fig. 4 depicts the partial process of the genetic algorithm. Four different sets of random parameters are used as the initial population. Each set of random parameters is called a chromosome, and each parameter is called a gene. The four chromosomes of the initial generation were ranked using a fitness function. The fitness function was defined as the sum of the ratios of the four performance indicators of RocksDB. The formula for the fitness function is as follows.

$$Fitness = 1000 \times \left[\sum_{\Gamma \in \{WAF, SA, TIME\}} \left(\frac{\Gamma_{default} - \Gamma_{chromosome}}{\Gamma_{default}} \right) + \left(\frac{RATE_{chromosome} - RATE_{default}}{RATE_{default}} \right) \right] \quad (2)$$

The best chromosome with the highest fitness value is then estimated to determine its sufficiency for this purpose. The genetic algorithm stops when the best chromosome meets this requirement. Two chromosomes with the highest and next-highest fitness values survive, whereas the chromosomes with the bottom-ranking fitness values are discarded. The top two surviving chromosomes become the parents of the next generation, and a new chromosome is generated by crossing them. In the meantime, a gene mutation occurs with a certain probability to insert a new gene, which allows to explore various chromosomes. Therefore, two parent chromosomes, one crossover chromosome, and one mutant chromosome become the chromosomes of the next generation. The next-generation chromosomes are ranked again using the fitness function, which means that the algorithm enters another iteration. Chromosomes with high fitness values continue to survive, and with each generation, the chromosome with the highest fitness value gets closer to the optimal solution.

4. Experiment

4.1 Environment and Data Generation

The experimental environments used in this study are listed in Table 1. db_bench was used to measure the performance of RocksDB [15]. It is a performance measurement tool provided by RocksDB and includes benchmarks for various parameters such as the RocksDB level size, compression technique, data size, and number. In this experiment, the performance of RocksDB was measured by loading 1,000,000 key-value pairs. The key size and value size were fixed at 16 B and 1 kB, respectively; 30 parameters related to WA and SA were selected from approximately 200 RocksDB parameters. These parameters had random values. Data for 2,000 random combinations were generated, and the size of the LSM-Tree was reduced to 1/64th that of the default size to efficiently shorten the data generation time.

Table 1. Experiment environment

	Specification
OS	CentOS 7.3.1611 (x86_64)
CPU	Intel Xeon CPU E5-2660 v2 @ 2.20GHz
RAM	Samsung M393B2G70QH0-YK 16 GB*4
NVMe	Intel SSDPEDMD800G4 DC P3700 800 GB
RocksDB	Version 6.13

4.2 Performance Indicators

We selected four performance indicators, namely WA factor (WAF), loaded data size (SA), data processing rate (RATE), and execution time (TIME), to analyze the changes in the performance of RocksDB based on various parameters. RATE and TIME were selected such that unacceptable values of RATE and TIME could be avoided when improving WAF and SA. Our purpose was to improve the overall performance. WAF is a performance indicator representing the ratio of the amount of data written to the storage and the amount of data written to the database. It is calculated as follows.

$$\text{WAF} = \frac{\text{Bytes written to storage}}{\text{Bytes written to database}}. \tag{3}$$

The loaded data size, also called SA, is measured by the size of the data recorded in the actual LSM-Tree [16]. Because an LSM-Tree contains both valid and invalid SST files, the size of data recorded in LSM-Tree can be used as an effective metric for the actual size of physical space exploited by RocksDB. The data processing rate refers to the number of operations processed by RocksDB per second, including compaction, compression, read, and write operations. Execution 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 size allocated to RocksDB increased; however, TIME remained the same even when the number of operations increased, owing to the large compression ratio.

4.3 Random Forest Learning Assessment

The results of the random forest-based analyses of the four performance indicators, namely, WAF, SA, RATE, and TIME, are depicted in Fig. 5. We used a random forest, implemented in scikit-learn [17]. We performed cross-validation to evaluate the performance of the random forest by dividing 2,000 samples

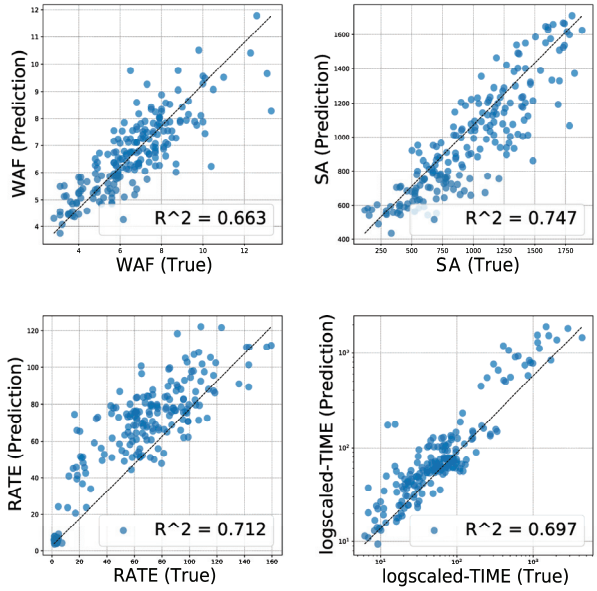

Fig. 5. Prediction accuracy for each RocksDB performance indicator.

into training and test samples in a ratio of 8:2. For the test samples, the coefficients of determination for predicting WAF, SA, RATE, and TIME were 0.663, 0.747, 0.712, and 0.697, respectively. Therefore, it can be said that the learning was effective. The trained random forest was continuously used for parameter contribution analysis.

4.4 Parameter Contribution Analysis

Fig. 6 depicts the results of the analysis of the parameter contribution based on the Gini average reduction. The results confirmed that the contribution of *min_level_to_compress* was the largest, followed by *compression_type*, *thread*, *level0_slowdown_writes_trigger*, *write_buffer_size*, *compression_*

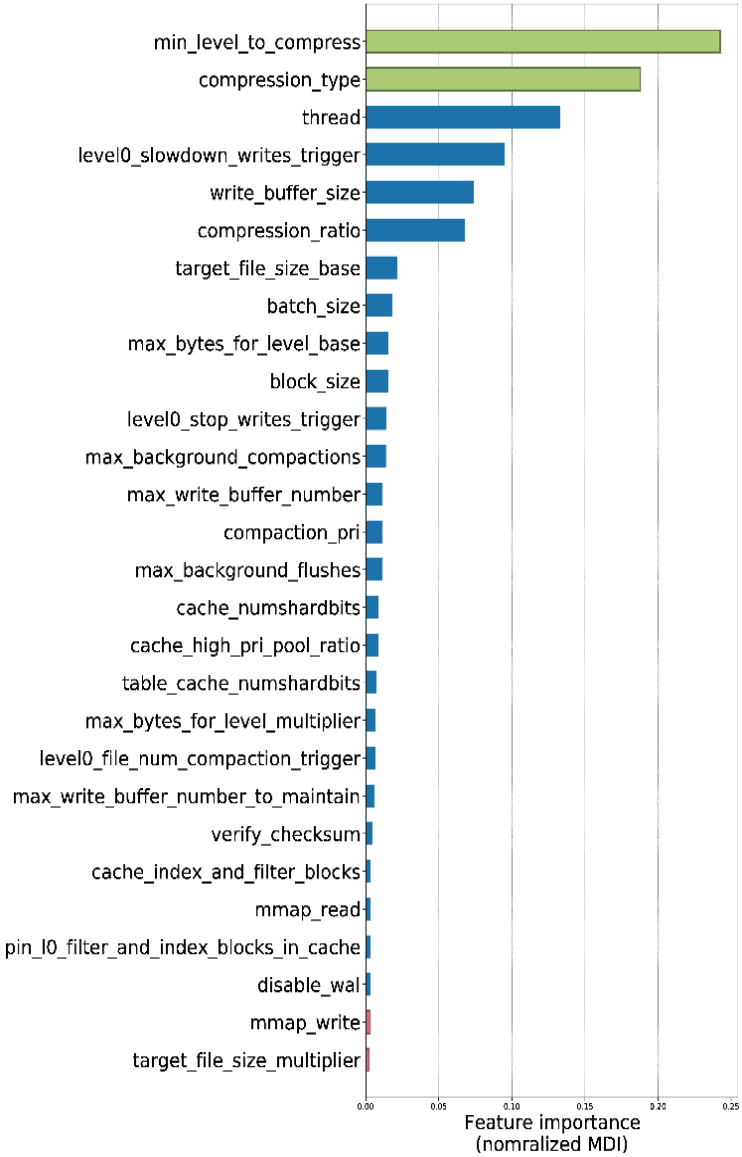

Fig. 6. Parameter contribution based on Gini average reduction (MDI=mean decrease impurity).

ratio, etc. The parameter *min_level_to_compress* configures RocksDB to perform compression from the same level as the parameter value. Levels lower than this parameter were not compressed. The default value was -1, and compression was performed at all levels. *compression_type* indicates the compression algorithm to be applied. The default algorithm is *snappy*; however, there are other algorithms such as *none*, *zlib*, *bzip2*, and *lz4*. The *thread* represents the number of threads that will execute the requested command; the default value is 1. *level0_slowdown_writes_trigger* controls the number of L0 files that slow down the write speed of RocksDB; the default value is 20. *write_buffer_size* is the number of bytes to be buffered in the RocksDB memtable before compression; the default value is 64 MB. The *compression_ratio* represents the data compression rate, and the default value is 0.5. In fact, *compression_type* is an important parameter that has been studied by various research teams to improve the performance of RocksDB [17].

Fig. 7 depicts the results of performing ANOVA by selecting two parameters from the top and bottom, respectively, based on the parameter contribution shown in Fig. 6. Fig. 7(a) and 7(b) show the difference in the performances of SA and RATE, based on *min_level_to_compress* and *compress_type,* corresponding to the highest parameter. When each performance difference was statistically tested through ANOVA, the p-values were 4.84e-31 and 8.53e-68, respectively, which were significantly smaller than the significance level of 0.05. This confirmed that the difference in performance was significant, based on *min_level_to_compress* and *compress_type*. Fig. 7(c) and 7(d) depict the results of

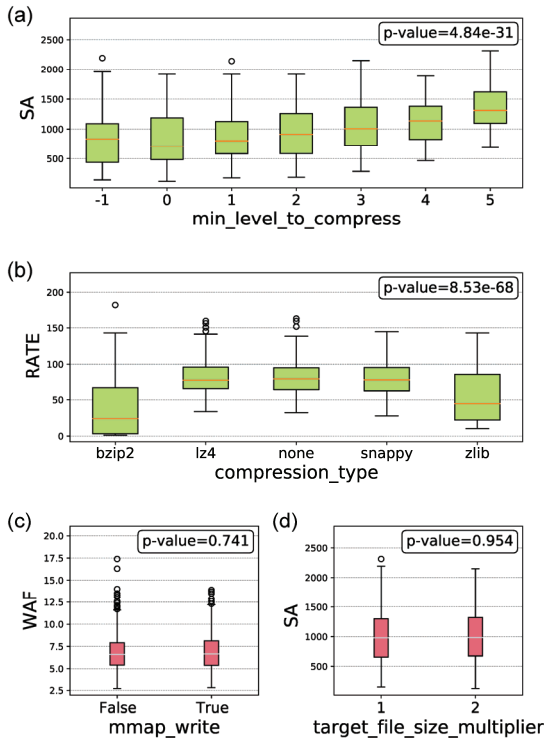

Fig. 7. Comparative analysis of the highest and lowest parameters. (a) and 7(b) show the difference in the performances of SA and RATE, based on *min_level_to_compress* and *compress_type*, corresponding to the highest parameter. (c) and 7(d) depict the results of analyzing the performance difference between WAF and SA based on *mmap_write* and *target_file_size_multiplier* corresponding to the lowest parameter.

analyzing the performance difference between WAF and SA based on *mmap_write* and *target_file_size_multiplier* corresponding to the lowest parameter. Unlike the result of the top parameter, the p-values were 0.741 and 0.954, respectively, indicating higher values than the significance level. Therefore, it was confirmed that the two parameters did not significantly contribute to the performance of RocksDB. In fact, when comparing the minimum (821.5) and maximum (1328) average SAs, based on the *min_level_to_compress* value, there was a significant difference in performance, which was 61.7%. Based on *compression_type*, the average RATE ranged from a minimum of 36.57 to a maximum of 80.16, which turned out to be important for optimizing RocksDB. In contrast, the average difference between the maximum and minimum WAF and SA for *mmap_write* and *target_file_size_multiplier*, which were the lowest parameters, were found to be 3.77% and 0.51%, respectively, indicating that they were not much relevant to the performance improvement of RocksDB.

4.5 Genetic Algorithm Optimization Solution

The top five parameters of RocksDB with the highest contribution were selected via random forest analysis. Among the selected parameters, *thread* was discarded because it changed based on the number of users. Finally, the genetic algorithm was applied to four parameters: *compression_type*, *level0_slowdown_writes_trigger*, *write_buffer_size*, and *compression_ratio*.

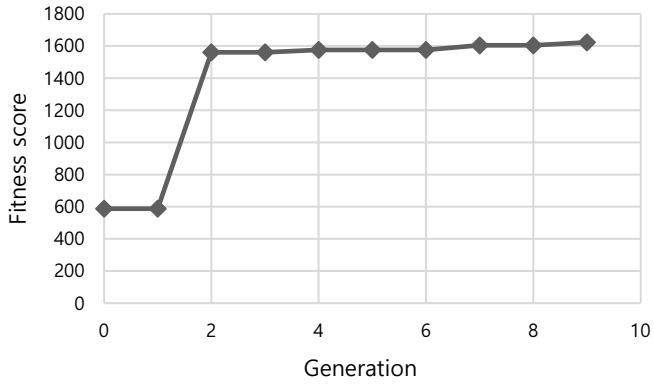

Fig. 8. Convergence process of genetic algorithm.

We defined eight random chromosomes as the initial population, and each generation always had eight chromosomes. The bottom four chromosomes were discarded in each iteration when the best chromosome did not satisfy this requirement. To refill four chromosomes, two chromosomes were generated by the crossover of parent chromosomes, without any mutations, and the remaining two chromosomes were generated via complete mutations (new chromosome). The two latter chromosomes were completely mutated to ensure that the algorithm was less affected by the initial population.

Fig. 8 shows the convergence process of the genetic algorithm. The algorithm did not have a high fitness score in the first two generations. However, the potentially good genes in the previous phases survived, resulting in a high fitness score of approximately 1600 at the 7th generation. We found that optimal values were found in a short time, which means that the proposed pipeline may be an efficient method of parameter tuning for RocksDB.

Table 2 lists the results of calculating the optimal values of the top four parameters using the genetic algorithm. It is evident that the database with optimal values of RATE, SA, and WAF achieved better performance than that with the default settings. Fig. 9 shows the degree of improvement in the default parameters and the optimal parameters obtained through the genetic algorithm. Although there was only a slight reduction (-5.64%) in TIME, parameters WAF, SA, and RATE improved significantly by 20.65%, 54.50%, and 89.68%, respectively. The results demonstrate that our method allows RocksDB to eliminate unnecessary write operations and space occupancy and to achieve high data processing rate without delayed execution time.

Table 2. Performance of genetic algorithm optimized parameters compared to that of default parameters

Indicator	TIME (s)	RATE (MB/s)	SA (MB)	WAF
Default setting	**63.8**	15.5	432.94	9.2
Genetic algorithm	67.4	**29.4**	**197**	**7**

Bold notation indicates better performance.

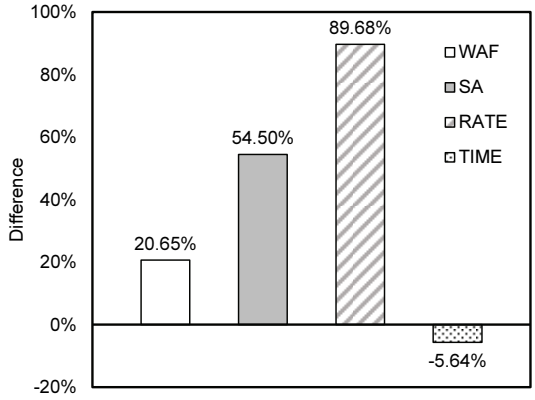

Fig. 9. RocksDB performance improvement optimized by the proposed tuning method.

5. Related Work

Database parameter tuning is a crucial task for improving database performance. The goal of database parameter tuning is to find the optimal parameters that can improve databases. Kanellis et al. [18] claimed that tuning a few parameters is sufficient for performance improvement on various workloads and database systems. OtterTune uses a combination of supervised and unsupervised machine learning methods to select the most influential parameters, to map previously unseen database workloads to known workloads, and to recommend parameter settings [19]. It first captures the features of the known workloads by applying factor analysis and k-means clustering. Then OtterTune uses lasso algorithm to identify the top-n important parameters. Finally, OtterTune uses gaussian process to find the best parameters. To map the workload correctly, OtterTune highly relies on data repository in the previous experience. BestConfig uses the divide-and-diverge sampling method and the recursive bound-and-search algorithm to divide parameter spaces into several subspaces [20]. CDBTune applies the deep

deterministic policy gradient method and the try-and-error strategy to learn the best configurations with limited samples [21]. Instead of traditional regression, CDBTune uses a reward feedback method to accelerate the speed of convergence and improve the performance of tuning. CDBTune has two main steps: offline training and online tuning. CDBTune first trains the model with standard benchmark and workloads in offline training, then apply the pre-trained model to tune the specific databases which is called online tuning. It takes several hours and about 20 minutes to finish the offline training and online tuning, respectively.

In this work, we proposed a parameter tuning method for RocksDB using a random forest and genetic algorithm. We built a tuning pipeline for RocksDB, where important parameters related to WAF, SA, TIME, and RATE were selected, and the optimal values of these parameters were explored by crossover and mutation techniques.

6. Conclusion

Database tuning and optimization are difficult tasks in the era of big data and modern technology. In this study, 30 parameters related to WA and SA, which are the major performance indicators of RocksDB, were selected from 200 RocksDB parameters. The effects of each parameter on WA and SA were analyzed using a random forest. The most important parameters are *compression_type*, *level0_slowdown_writes_trigger*, *write_buffer_size*, and *compression_ratio*. We found that these findings were consistent with the results of other parameter analysis studies, and ANOVA showed that the identified parameters were highly involved in the WA and SA performances. It is expected that the proposed method can identify influential parameters among numerous RocksDB parameters and determine the optimal configuration to improve the performance of RocksDB. The identified parameters can also be considered for other databases using LSM-Tree, which means that they play important roles in other database systems. Although parameter names may vary across database systems, their functions are the same.

We applied a genetic algorithm to successfully optimize the identified parameters and reduce WA and SA. The results indicated that WAF, SA, and RATE can be significantly improved by 20.65%, 54.50%, and 89.68%, respectively; however, there was only a minor reduction in TIME (-5.64%). WA and SA are the major factors that slow down the database and shorten the lifespan of storage devices. By optimizing the identified parameters, it is expected that cost-effectiveness and long-term use of storage devices can be achieved by alleviating unnecessary data writing and space occupancy.

In the future, we plan to perform parameter optimization experiments on other databases that use LSM-Tree and a database study that can be automatically tuned by analyzing the correlation of parameters using reinforcement learning techniques. The proposed method has a critical limitation that requires a long time for the parameter analysis. To elicit the performance of random forests, we generated 2,000 random configurations using db_bench with approximately 200 RocksDB parameters. Increasing the size of training data improves the tuning quality but requires excessive time for data generation. Moreover, parameter optimization with a genetic algorithm also consumes considerable amounts of time for checking various chromosomes and fitness scores. To overcome this limitation, we plan to study deep learning that can perform both feature selection and optimal value prediction simultaneously. In the

future, we will develop deep learning-based tuning methods to reduce the running times for training data preparation and parameter optimization tasks.

Acknowledgement

This work was supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. 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) and Korea Ministry of Land, Infrastructure and Transport (MOLIT) as “Innovative Talent Education Program for Smart City”.

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, 2018.

[2] RocksDB, “A persistent key-value store for fast storage environments,” 2021 [Online]. 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,” in *Proceedings of the International Conference on Compute and Data Analysis*, Lakeland, FL, 2017, pp. 155-161.

[5] Z. Cao, S. Dong, S. Vemuri, and D. H. Du, “Characterizing, modeling, and benchmarking RocksDB key-value workloads at Facebook,” in *Proceedings of the 18th USENIX Conference on File and Storage Technologies (FAST 2020)*, Santa Clara, CA, 2020, pp. 209-223.

[6] H. Kim, J. H. Park, S. H. Jung, and S. W. Lee, “Optimizing RocksDB for better read throughput in blockchain systems,” in *Proceedings of the 23rd International Computer Science and Engineering Conference (ICSEC)*, Phuket, Thailand, 2019, pp. 305-309.

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

[8] Y. Lu, J. Shu, and W. Zheng, “Extending the lifetime of flash-based storage through reducing write amplification from file systems,” in *Proceedings of the 11th USENIX Conference on File and Storage Technologies (FAST 2013)*, San Jose, CA, 2013, pp. 257-270.

[9] S. Odeh and Y. Cassuto, “NAND flash architectures reducing write amplification through multi-write codes,” in *Proceedings of 2014 30th Symposium on Mass Storage Systems and Technologies (MSST)*, Santa Clara, CA, 2014, pp. 1-10.

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

[11] D. C. Howell, *Statistical Methods for Psychology*. Pacific Grove, CA: Duxbury, 2002.

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

[13] 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.

[14] 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.

[15] GitHub, "Benchmarking tools," 2022 [Online]. Available: https://github.com/facebook/rocksdb/wiki/Benchmarking-tools.

[16] S. Dong, M. Callaghan, L. Galanis, D. Borthakur, T. Savor, and M. Strum, "Optimizing space amplification in RocksDB," in *Proceedings of the 8th Biennial Conference on Innovative Data Systems Research (CIDR)*, Chaminade, CA, 2017.

[17] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, et al., "Scikit-learn: machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825-2830, 2011.

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

[19] D. Van Aken, A. Pavlo, G. J. Gordon, and B. Zhang, "Automatic database management system tuning through large-scale machine learning," in *Proceedings of the 2017 ACM International Conference on Management of Data*, Chicago, IL, 2017, pp. 1009-1024.

[20] Y. Zhu, J. Liu, M. Guo, Y. Bao, W. Ma, Z. Liu, K. Song and Y. Yang, "BestConfig: tapping the performance potential of systems via automatic configuration tuning," in *Proceedings of the 2017 Symposium on Cloud Computing*, Santa Clara, CA, 2017, pp. 338-350.

[21] J. Zhang, Y. Liu, K. Zhou, G. Li, Z. Xiao, B. Cheng, 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*, Amsterdam, Netherlands, 2019, pp. 415-432.

Huijun Jin **https://orcid.org/0000-0003-2470-6676**

He received his B.S. degree in computer science from Yanbian University of Science and Technology, Yanji, China, in 2018. He is currently pursuing his M.S. degree at the Data Engineering Lab of the Department of Computer Science, Yonsei University, Seoul, South Korea. His current research interests include database systems, big data, key-value stores, and machine learning.

Won Gi Choi **https://orcid.org/0000-0001-7793-4651**

He received his B.S. degree and Ph.D. degree in Computer Science from Yonsei University in 2014 and 2021. He is currently a researcher at Korea Electronics Technology Institute (KETI). His current research interests include database systems, digital twin, and big data platforms.

Jonghwan Choi **https://orcid.org/0000-0002-8429-4135**

He received his B.S. degree in 2016 from the Department of Mathematics and M.S. degree in 2018 from the Department of Computer Science & Engineering, Incheon National University, Incheon, South Korea. He is currently pursuing Ph.D. degree at the Department of Computer Science, Yonsei University, Seoul, South Korea. His current research interests include bioinformatics, machine learning, deep learning, and data mining.

Hanseung Sung **https://orcid.org/0000-0003-4797-9060**

He received his B.S. degree in software and computer engineering from Korea Aerospace University in 2017. He received his M.S. degree in computer science from Yonsei University in 2020. He is currently working as a Post-Master's Researcher in the Data Engineering Lab of the Department of Computer Science, Yonsei University, Seoul, South Korea. His current research interests include database systems, in-memory, key-value stores, and logging and recovery.

Sanghyun Park **https://orcid.org/0000-0002-5196-6193**

(Member, IEEE) He received his B.S. and M.S. degrees in computer engineering from Seoul National University in 1989 and 1991, respectively, and Ph.D. from the Department of Computer Science, University of California at Los Angeles (UCLA), in 2001. He is currently a Professor with the Department of Computer Science, Yonsei University, Seoul, South Korea. His current research interests include databases, data mining, bioinformatics, and flash memory.