Tuning Cassandra through Machine Learning

Florian Eppinger[1] Uta Störl[2]

Abstract: The distributed nature and scalability of NoSQL databases make them an ideal data storage repository for a variety of use cases. While NoSQL databases are delivered with a default "off-the-shelf" configuration, they offer configuration settings to adjust a database's behavior to a specific use case and environment. The abundance and oftentimes imperceptible inter-dependencies of configuration settings make it difficult to optimize and performance-tune a NoSQL system. This work explores Machine Learning as a means to automatically tune a NoSQL database for optimal performance. Using Ensemble Machine Learning algorithms, multiple ML models were fitted with a training dataset that incorporates properties of the NoSQL physical configuration (replication and sharding). The best models were then employed as surrogate models to optimize the Database Management System's configuration settings for best performance using a Black-box Optimization algorithm. Multiple experiments were carried out with an Apache Cassandra database to demonstrate the feasibility of this approach, even across varying physical configurations. The tuned Database Management System configurations yielded throughput improvements of up to 4%, read latency reductions of up to 43%, and write latency reductions of up to 39% when compared to the default configuration settings.

Keywords: AI for Database Systems; NoSQL; Machine Learning; Performance Modeling; Tuning; Black-box Optimization

1 Introduction

The rate at which data is created, used, and persisted increases rapidly. While Relational Database Management Systems (RDBMSs) continue to play an important role in today's technology environments, Non-relational SQL or Non-SQL (NoSQL) Databases (DBs) have become an integral part of real-time analytics or big data applications [CL19; SF12]. Choosing the best NoSQL solution and developing the best physical design for a given use case can be a challenging task on its own [HAR16; Lo15; QCH18]. Furthermore, NoSQL technologies offer an abundance of configuration settings that allow the system administrator to adjust the DB behavior to further meet the requirements of a particular use case. Many of the configuration parameters have an impact on the performance of the NoSQL DB, i.e., its throughput and latency.

Finding the configuration that maximizes throughput or minimizes latency for a given use case is complex, and DB behavior is not always self-explanatory as shown by Preuveneers; Joosen [PJ20].

[1] University of Hagen, Databases and Information Systems, Hagen, Germany florian.eppinger@gmail.com
[2] University of Hagen, Databases and Information Systems, Hagen, Germany uta.stoerl@fernuni-hagen.de

doi:10.18420/BTW2023-04

Having the ability to predict performance measures for varying workloads and physical configurations, and to optimize Database Management System (DBMS) configuration settings accordingly, could be beneficial due to a variety of reasons, including sudden changes in user behavior, application changes, and changes to the hardware environment.

The methodology and results outlined in this study consider aspects of the physical configuration *and* the DBMS configuration settings during tuning, and we thus make the following contributions:

- We analyze the quality of Machine Learning (ML) models that are trained to predict performance metrics for varying workloads, physical configurations, and DBMS configurations.
- We measure the influence of the training dataset size on the quality of these ML models.
- We evaluate how features representing the physical configuration impact the quality of the ML models.
- We explore the ability to tune the DBMS configuration settings for a specific physical configuration using the ML models and Black-box Optimization (BBO).

Section 2 reviews related work. Section 3 briefly introduces the end-to-end methodology and the training dataset. Results and findings are presented and evaluated in section 4. The document concludes with section 5, which discusses the results and suggests areas of future work.

2 Related Work

A variety of proposals have been made to mitigate the performance tuning challenges outlined in section 1 through automation. Some of these methods are able to tune a variety of software solutions in a generic fashion [Wa18; Zh17], while others are geared specifically toward RDBMS or NoSQL technology. This work can be categorized into solutions that tune DB performance via the DBMS configuration settings [Ak21; KAS15; Xi17; Zh19] and solutions that tune DB performance via the physical design of the DB [Be15; Cr13; Fa16]. Tuning Methods include Control Theory, Expert Systems, and a variety of Machine Learning algorithms.

The main difference between this paper and related performance-tuning work for NoSQL systems is that the Tuning Domain includes both DBMS configuration settings *and* the DB physical design in form of sharding and replication. While Preuveneers and Joosen consider both aspects [PJ20], their technique differs in several ways. First, Preuveneers and Joosen utilize multiple predefined tactics in an attempt to tune the NoSQL system *online*. Second, the authors utilize the ML model to map DB scenarios consisting of workload metrics, resource utilization, physical configuration, etc. to an ideal DBMS configuration and DB physical design. On the other hand, this study evaluates ML techniques to fit ensemble models to

predict DB performance. These models are then used by BBO algorithms to find an optimum DBMS configuration for a given workload and physical configuration. Consequently, the methods used in this paper are related much closer to the approach presented by Xiong et al. [Xi17], except that this work includes features for the physical configuration, utilizes Apache Cassandra instead of HBase, and employs a different optimization algorithm. For an in-depth study of related work and a point-by-point comparison refer to the extended version of this paper [ES22].

3 Methodology

Apache Cassandra[3] ("Cassandra") was selected as the basis for the experiments. A tuning domain was defined, and a training dataset was generated using a sample database. This training dataset was then transformed into input for Decision Tree (DT) ML algorithms to fit various ML models. The models' objective is to accurately predict performance metrics of the database for a given workload, DBMS configuration, and physical design. The performance of the ML models was evaluated using commonly accepted quality measures. The predictions of the ML models were then used to find optimized DBMS configuration values for a given workload and physical configuration using a Black-box Optimization algorithm. Finally, actual performance of the optimized configurations was evaluated and compared against the DB performance of the default DBMS configuration.

3.1 Tuning Domain

This section describes the Tuning Domain (TD), i.e., the workload properties, the specific DBMS configuration settings, as well as aspects of the Cassandra physical configuration that were considered within the scope of this study:

- *Workload:* To follow the approach chosen in related work, such as [Cr13] and [PJ20], three workloads were included:
  - 50% read using the primary key, 50% write (*readwrite*)
  - 95% read using the primary key, 5% write (*readyheavy*)
  - 5% read using the primary key, 95% write (*writeheavy*)
- *DBMS Configuration:* To reduce the complexity of this study, a subset of performance-relevant Apache Cassandra configuration settings was selected through research and targeted experiments.
- *Physical Design:* to account for aspects of the physical configuration, we considered both *sharding* and *replication*.

Table 1 provides a summary of the features and feature domains that were considered in this study.

[3] https://cassandra.apache.org/ (visited on Sept. 19^{th}, 2022)

Feature	Feature Category	Domain
wl_read_%	Workload	(0% – 100%)
wl_write_%	Workload	(0% – 100%)
trickle_fsync	DBMS Configuration	{true, false}
key_cache_size_in_mb	DBMS Configuration	$\{0, 2^0, ..., 2^5\}$
row_cache_size_in_mb	DBMS Configuration	$\{0, 20, 40, 60, ..., 200\}$
commitlog_segment_size_in_mb	DBMS Configuration	$\{2^2, ..., 2^6\}$
concurrent_reads	DBMS Configuration	$\{2^1 * disks, ..., 2^5 * disks\}$
concurrent_writes	DBMS Configuration	$\{2^1, ..., 2^8\}$
memtable_heap_space_in_mb	DBMS Configuration	$\{2^{-5} * heap, ..., 2^{-1} * heap\}$
node_count (n)	Physical Design	$\{2, 3, 4\}$
replication_factor (rf)	Physical Design	$\{1, 2, 3, 4\}$

Tab. 1: Features included in the Tuning Domain

3.2 Tuning Subdomains

In this study, we analyzed the impact of the workload, DBMS configuration, and the physical design on the accuracy of the ML models and their ability to serve as a surrogate model for performance tuning. We, therefore, defined TD1-TD4 as subsets of the TD. As shown in table 2, these *Tuning Subdomains* focus on specific aspects of the feature set.

TD	Physical Config.	Workload	DBMS Config.
TD1	*	*	*
TD2	*	wl_read_%=fixed, wl_write_%=fixed	*
TD3	n=fixed, rf=fixed	*	*
TD4	n=fixed, rf=fixed	wl_read_%=fixed, wl_write_%=fixed	*

Tab. 2: Definition of the tuning subdomains

TD1 represents the entire tuning domain, i.e. all of the features introduced in Table 1. TD2 reduces complexity by removing the workload features from the feature vector. It was designed to train ML models that are able to make predictions for a fixed workload. TD3, on the other hand, excludes the physical configuration features from the feature vector. TD4 only considers the DBMS configuration settings.

3.3 Tuning Methodology

The methodology applied in this study used ensemble-based ML algorithms to develop models that are able to make DB throughput and latency predictions. Leveraging the ML models' predictions, a BBO algorithm is then utilized to search the optimum DBMS configuration for a given workload and DB physical configuration.

Ensemble methods combine multiple ML models to make a final prediction. Two popular ensemble methods are *Bootstrap Aggregation (Bagging)* [Br01] and *Boosting* [Fr01; NK13], which were explored in form of Random Forest (RF) and Gradient Boosting Decision Tree (GBDT) ML algorithms for various reasons that are outlined in the extended version of this paper [ES22].

The fitted ML model can be considered a surrogate model that can respond with a performance prediction for a given set of inputs. Because the model itself is a black-box and does not expose any meaningful information that could be used to find an optimum using a gradient-based optimization approach, this study utilized BBO as a means to find an optimum configuration using the ML model's performance predictions. Various BBO algorithms were explored. However, the results presented in this paper are based on the Simulated Annealing algorithm, which was shown to be the most efficient among the ones evaluated [ES22]. The algorithm is similar to Local Optimization except that it attempts to avoid getting stuck in a non-global optimum by adding a random element for further exploration of the domain [Jo89].

3.4 Training Data

A new dataset consisting of 32,757 training examples was created. A training example comprises workload properties, the values for DBMS configuration settings, a representation of the DB physical configuration, as well as the actual performance values that were measured when the workload was executed against the DB using this particular DBMS and physical configuration. Additional details about the training dataset can be found in the extended version of this study [ES22].

4 Experimental Evaluation

The following results and findings were gathered during experiments that were carried out on a DB cluster with 5 nodes provided by the University of Hagen. Please refer to the extended version of this paper [ES22] for additional details regarding the implementation as well as a more in-depth analysis of results beyond the DB read and write latencies that this paper focuses on.

4.1 ML Model Quality

Table 3 lists the Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE) for DB throughput and latencies. The measurements were established by training a total of 6 individual ML models using TD1, i.e. the entire training dataset as defined in section 3.2.

	Overall Throughput		Read Latency		Write Latency	
	RF	GBDT	RF	GBDT	RF	GBDT
MAE	2,810.940	2,880.110	0.307	0.279	0.203	0.189
MAE (%)	3.290	3.370	2.940	2.730	3.030	2.890
RMSE	4,646.420	5,064.020	0.500	0.493	0.369	0.369

Tab. 3: ML Model Quality

It should be noted that the hyperparameters of the ML models referenced in this study were tuned for each ML model individually. Using a random split methodology, 75% of the dataset was used to fit the models with holdout validation (see [Gé17]), and the remaining 25% were used for testing.

To understand the correlation between dataset size and ML model prediction accuracy, various experiments were conducted with artificially reduced datasets. A total of 9 ML models were trained for each of the performance measures using the following dataset sizes: 128, 256, 512, 1,024, 2,048, 4,096, 8,192, 16,384, 24,672. To ensure a fair evaluation and reduce the chance of randomly selecting a non-representative test distribution, the test dataset was kept at a fixed size of 8,000 examples. The results are shown in figure 1. A training dataset size of 128 examples yielded a GBDT model that predicted overall throughput with an MAE of 7,326 and a RF model that predicted overall throughput with an MAE of 9,973. Increasing the training dataset size to just 1,024 records significantly reduced the MAE values to 3,903 (GBDT) and 4,305 (RF). Additional accuracy could be gained by further increasing the dataset size. However, only minor improvements could be seen for read and write latencies with datasets exceeding 4,096 examples.

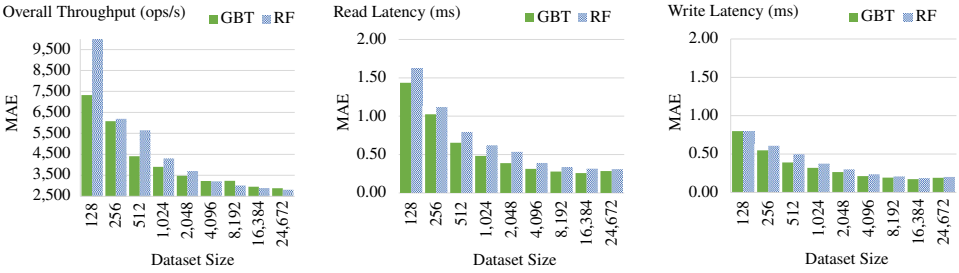

Fig. 1: Influence of the dataset size on the prediction accuracy of the ML models

To compare the TD1 ML model quality to the less complex Tuning Domains, ML models were trained for TD2-TD4 to predict write latencies. Four individual models were fitted for each combination of TD and ML algorithm using datasets with 128, 256, 512, and 1,024 training examples. For this experiment, a test dataset size of 250 was used to determine the MAE values for each of the models. The results are shown in figure 2 and confirm that the ML models trained with TD1 performed worse compared to TD2-TD4 when identically sized training datasets were used. A GBDT model trained with 128 randomly

selected examples from the TD1 dataset yielded an MAE of 1.42, which represents an error percentage of 13.62% compared to 0.65 (5.88%) for TD2. The MAE of the GBDT model dropped to 0.39 (3.55%) with 512 training examples for TD2, while it took four times the number of training examples to reach comparable accuracy within TD1. It is also worth noting that with 1,024 training examples, the TD1 and TD3 models were of similar quality despite the added complexity of the TD1 model that is able to make predictions for eight different physical configurations.

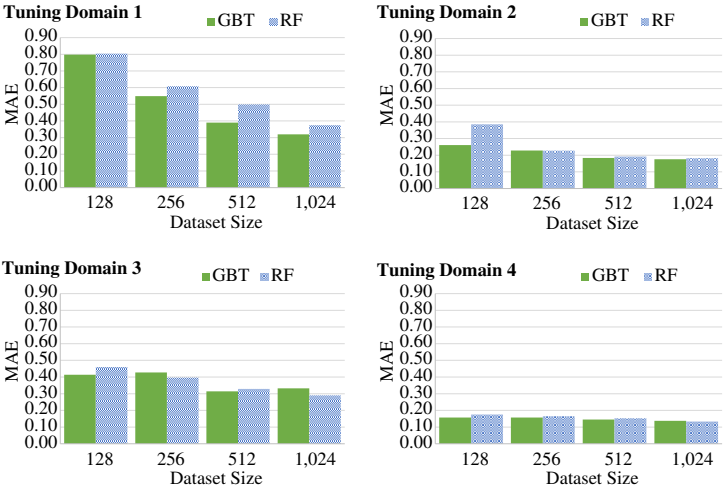

Fig. 2: Influence of the dataset size on the write latency prediction accuracy of the ML models trained with different Tuning Domains

4.2 Tuning performance with Black-box Optimization

The Simulated Annealing BBO algorithm was then used to optimize the DBMS configuration settings in a variety of experiments that involved the fitted ML models. The configurations proposed by the algorithm were then applied to the Cassandra cluster, workloads were executed, and results were captured.

First, we optimized DBMS configuration settings for various workloads with a fixed physical configuration (n=4, rf=3) using the fitted TD1 model. In addition to the *readwrite* (50% read, 50% write), *readheavy* (95% read, 5% write) and *writeheavy* (5% read, 95% write) workloads, experiments were conducted with a workload that differed from the workloads used to train the ML model (25% read, 75% write). For each performance measure, five independent experiments were carried out for each proposed DBMS configuration and workload, and the average performance was calculated. Results for the actual read and write latencies are shown in figure 3. Tuned configurations reduced the read latency by more than 42% for write-heavy workloads and the write latency by more than 39% for workloads with a significant amount of read operations. It should be noted that configurations that improved read latency resulted in a higher write latency and vice versa, reducing overall throughput.

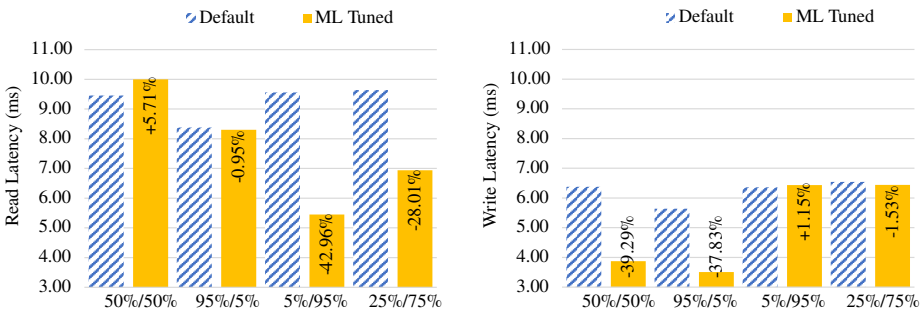

Fig. 3: Actual latency measurements for default and ML/BBO-tuned DBMS configurations

However, the results clearly demonstrate that the ML model was able to successfully derive information regarding the DBMS configuration settings' impact on the DB performance.

Next, we applied the optimization method to different physical configurations. This time the objective of the algorithm was to tune the performance by adjusting the DBMS configuration for varying physical configurations. The measurements also included a physical configuration with two nodes and a replication factor of one to analyze how well the methodology works for previously unseen physical configurations. Figure 4 compares the write latency of the ML-tuned configuration to the default configuration (*readwrite* workload). The results demonstrate that the ML-based tuning method successfully tuned the DB latency for a variety of physical configurations. However, it also highlights that it failed to optimize

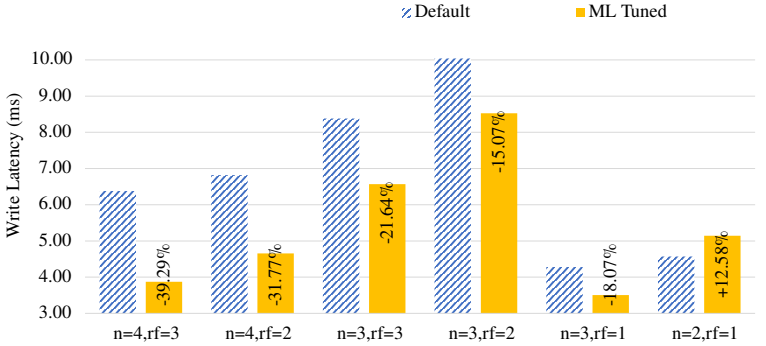

Fig. 4: Write latencies for default and ML/BBO-tuned DBMS configurations under varying physical configurations

performance for the physical configuration that was not previously encountered during the training phase. The write latency increased by 12.58% from 4.57*Milliseconds*(ms) to 5.14ms. It is possible that the ML algorithms failed to extract and derive a meaningful trend or that the ordinal encoding method for the corresponding features did not cultivate this kind of predictive quality in the model.

We also analyzed models that were trained with the tuning subdomains TD2-TD4. Section 4.1

showed significant accuracy differences depending on what tuning domain was used to fit the ML models. To evaluate the ability to tune the DBMS configurations with these models, the Simulated Annealing algorithm was used to search for an optimum DB configuration for each of the TDs using the ML models fitted with 1,024 training examples. The actual DB performance was then measured for the proposed configurations. This process was repeated three times for each TD, and average performance results were calculated. The goal of the experiment was to optimize read and write latency for the *readheavy* workload (95% read/5% write). The results are shown in figure 5.

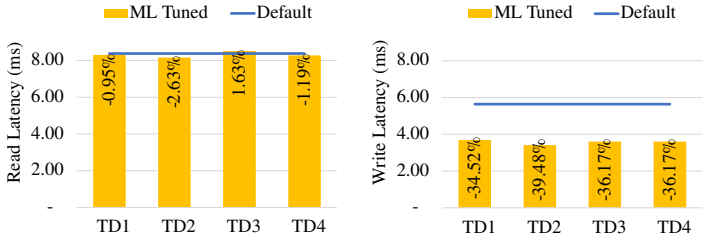

Fig. 5: Actual latency measurements for DBMS configurations that were optimized with ML models trained with TD1-TD4

The superior accuracy of TD2 and TD4 models did not result in better DBMS configuration proposals. We determined that the model's tuning ability depended both on its accuracy as well as the quality of the training data. Because the training subset was chosen randomly, it contained examples with poor performance. The ML models for the less complex TDs were able to make more accurate predictions, but this did not help the BBO algorithm find a better configuration if they were not fitted with high-performance training examples.

5 Discussion

This study evaluated Machine Learning and Black-box Optimization as a means to performance-tune NoSQL DBs for varying physical configurations and workloads. When fitted with the entire dataset, the ML models yielded an MAE of 2.73% and 2.89% for read and write latencies, respectively. Omitting the workload from the feature set (TD2) significantly improved accuracy, while omitting the features representing the physical configuration (TD3) resulted in moderate improvements only. This observation may lead to the conclusion that *the physical configuration adds less complexity to the model than varying workloads*. When using training datasets of 1,024 examples, the MAE of the TD1 and TD3 GBDT models were almost identical when predicting read latencies, thereby implying that it is more efficient to train a single predictive model for multiple physical configurations than it is to train an individual model for each physical configuration.
The most accurate models were then used to optimize DBMS configuration settings for a given workload and physical configuration using BBO. The algorithm was able to find configurations for improved performance in most situations. For a physical design with four NoSQL nodes and a replication factor of three, latencies could be reduced by up to

42.96% (read) and 39.29% (write) depending on the workload composition. Similar results were achieved for varying physical configurations. Additional experiments showed that DB latency could be improved even for previously unseen workloads.
While the tuned DBMS configurations did result in performance improvements, none of the results matched or exceeded the performance of the best-performing training examples. As an example, the BBO-tuned DBMS configuration (n=4, rf=3, writeheavy) resulted in maximum throughput of 103,965 operations per second (op/s) compared to 95,240 op/s for the default configuration. However, the most performant training example resulted in 111,018 op/s, implying an additional tuning potential of 6.5%. A potential explanation for this is the generalization capability of the ML model, which enables the ML model to make more accurate predictions for previously unseen DBMS configurations but also regulates outlier configurations. These outliers represent subpar and also optimum configurations.

Several discoveries and choices were made regarding technology and methodology. Furthermore, several areas remained unexplored due to time and resource constraints. The following list reflects on some of these items and highlights potential areas for future work:

- This study targeted individual DB performance measures. Performance objectives vary, and it may be necessary to meet multiple performance goals. Xiong et al. approach this by combining multiple weighted performance measures into a single optimization objective [Xi17], an attempt that could be explored further.
- It was also noted that the training dataset captures performance results that are specific to the hardware environment and technologies used. Exploring the transformation or scaling of training examples or derived knowledge for a different environment or technology stack appears worthwhile.
- The tuning domain of this study is limited. More aspects exist, including indexes, schema design, consistency levels, etc., that could be evaluated in more detail.
- Attempting to tune the DBMS configuration for previously unseen physical designs did result in performance that under-performed the default DBMS configuration. The corresponding features were encoded as ordinal values, and changing the encoding scheme may improve these results.
- The study treated DBMS configuration settings as global settings, i.e., the same configuration settings were used for all nodes of the Cassandra node ring. However, many settings can be configured individually for each cluster node. Research in this area presented by Cruz et al. [Cr13] could be evaluated as an extension to the methodology outlined in this document.
- This study evaluated RF and GBDT to develop a surrogate model for DB performance predictions. Other ML algorithms exist and may exhibit a better prediction quality. Similarly, various BBO algorithms exist, some of which have been shown to produce better results than the Simulated Annealing algorithms [Ch21]. A more systematic evaluation of different algorithms could be carried out.

References

[Ak21] Aken, D. V.; Yang, D.; Brillard, S.; Fiorino, A.; Zhang, B.; Bilien, C.; Pavlo, A.: An Inquiry into Machine Learning-based Automatic Configuration Tuning Services on Real-World Database Management Systems. Proc. VLDB Endowment 14/, pp. 1241–1253, 2021.

[Be15] Bermbach, D.; Müller, S.; Eberhardt, J.; Tai, S.: Informed Schema Design for Column Store-Based Database Services. In: Proc. SOCA 2015. IEEE, pp. 163–172, Oct. 2015.

[Br01] Breiman, L.: Random Forests. Machine Learning 45/, pp. 5–32, Jan. 2001.

[Ch21] Chen, P.; Huo, Z.; Li, X.; Dou, H.; Zhu, C.: ConfAdvisor: An Automatic Configuration Tuning Framework for NoSQL Database Benchmarking with a Black-box Approach. In: Bench 2020, Revised Selected Papers. Vol. 12614, Springer, pp. 106–124, 2021.

[CL19] Chen, J. K.; Lee, W. Z.: An introduction of NoSQL databases based on their categories and application industries. Algorithms 12/, May 2019.

[Cr13] Cruz, F.; Maia, F.; Matos, M.; Oliveira, R.; Paulo, J.; Pereira, J.; Vilaça, R.: MeT: Workload aware elasticity for NoSQL. In: Proc. EuroSys 2013. Pp. 183–196, 2013.

[ES22] Eppinger, F.; Störl, U.: NoSQL Database Tuning through Machine Learning. CoRR abs/2212.12301/, 2022, arXiv: 2212.12301, URL: http://arxiv.org/abs/2212.12301.

[Fa16] Farias, V. A.; Sousa, F. R.; Maia, J. G.; Gomes, J. P.; MacHado, J. C.: Machine Learning Approach for Cloud NoSQL Databases Performance Modeling. In: Proc. CCGrid 2016. IEEE, pp. 617–620, July 2016.

[Fr01] Friedman, J. H.: Greedy Function Approximation: A Gradient Boosting Machine. Source: The Annals of Statistics 29/, pp. 1189–1232, 2001.

[Gé17] Géron, A.: Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems. O'Reilly Media, Inc., 2017.

[HAR16] Herrero, V.; Abelló, A.; Romero, O.: NOSQL design for analytical workloads: Variability matters. In: Proc. ER 2016. Vol. 9974, Springer, pp. 50–64, 2016.

[Jo89] Johnson, D. S.; Aragon, C. R.; Mcgeoch, L. A.; Schevon, C.: Optimization By Simulated Annealing: An Experimental Evaluation; Part I, Graph Partitioning. Oper. Res. 37/, pp. 865–892, 1989.

[KAS15] Khattab, A.; Algergawy, A.; Sarhan, A.: MAG: A performance evaluation framework for database systems. Knowledge-Based Systems 85/, pp. 245–255, Sept. 2015.

[Lo15] Lourenço, J. R.; Cabral, B.; Carreiro, P.; Vieira, M.; Bernardino, J.: Choosing the right NoSQL database for the job: a quality attribute evaluation. Journal of Big Data 2/, Dec. 2015.

[NK13] Natekin, A.; Knoll, A.: Gradient boosting machines, a tutorial. Frontiers in Neurorobotics 7/, 2013.

[PJ20] Preuveneers, D.; Joosen, W.: Automated configuration of NoSQL performance and scalability tactics for data-intensive applications. Informatics 7/, Aug. 2020.

[QCH18] Qader, M. A.; Cheng, S.; Hristidis, V.: A comparative study of secondary indexing techniques in LSM-based NoSQL databases. In: Proc. SIGMOD 2018. ACM, pp. 551–566, May 2018.

[SF12] Sadalage, P. J.; Fowler, M.: NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence. Addison-Wesley Professional, 2012.

[Wa18] Wang, S.; Li, C.; Hoffmann, H.; Lu, S.; Sentosa, W.; Kistijantoro, A. I.: Understanding and auto-adjusting performance-sensitive configurations. In: Proc. ASPLOS 2018. Vol. 53, ACM, pp. 154–168, Mar. 2018.

[Xi17] Xiong, W.; Bei, Z.; Xu, C.; Yu, Z.: ATH: Auto-Tuning HBase's Configuration via Ensemble Learning. IEEE Access 5/, pp. 13157–13170, June 2017.

[Zh17] Zhu, Y.; Liu, J.; Guo, M.; Bao, Y.; Ma, W.; Liu, Z.; Song, K.; Yang, Y.: BestConfig: Tapping the performance potential of systems via automatic configuration tuning. In: Proc. SoCC 2017. ACM, pp. 338–350, Sept. 2017.

[Zh19] Zhang, J.; Liu, Y.; Zhou, K.; Li, G.; Xiao, Z.; Cheng, B.; Xing, J.; Wang, Y.; Cheng, T.; Liu, L.; Ran, M.; Li, Z.: An End-to-End Automatic Cloud Database Tuning System Using Deep Reinforcement Learning. In: Proc. SIGMOD 2019. ACM, pp. 415–432, June 2019.