

NoSQL Database Tuning through Machine Learning

Florian Eppinger
University of Hagen
Hagen, Germany
florian.eppinger@gmail.com

Uta Störl
University of Hagen
Hagen, Germany
uta.stoerl@fernuni-hagen.de

Abstract

NoSQL databases have become an important component of many big data and real-time web applications. Their distributed nature and scalability 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 and performance 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. There is no one-size-fits-all configuration and therefore the workload, the physical design, and available resources need to be taken into account when optimizing the configuration of a NoSQL database. This work explores Machine Learning as a means to automatically tune a NoSQL database for optimal performance. Using Random Forest and Gradient Boosting Decision Tree Machine Learning algorithms, multiple Machine Learning 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 throughput and latency using a Black-box Optimization algorithm. Using an Apache Cassandra database, multiple experiments were carried out to demonstrate the feasibility of this approach, even across varying physical configurations. The tuned Database Management System (DBMS) 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.

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 [SF12, CL19]. Choosing the best NoSQL solution and developing the best physical design for a given use case can be a challenging task on its own [LCC⁺15, HAR16, 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. For example, one would assume that if a NoSQL DB is deployed on a significantly more powerful hardware configuration, its performance will increase accordingly. However, Preuveneers and Joosen have shown that this is not necessarily the case through research that demonstrates that moving a MongoDB DB instance from low-end desktop hardware to high-end server hardware did not yield the expected results [PJ20].

Having the ability to predict performance measures for varying workloads and physical configurations, and to optimize DBMS configuration settings accordingly, could be beneficial in various situations, e.g.:

- Sudden spikes in user activity or processing needs may require quick action from Database Administrators (DBAs). Intuition may lead to acquiring additional hardware resources, resulting in an oftentimes extensive implementation process. Having the ability to quickly evaluate the performance of various physical configurations, such as temporarily reducing the replication factor or increasing the node count, could significantly improve the ability to counteract the spike and comply with Service-Level Agreements (SLAs).
- As an application matures, changes, or grows, the workload composition may also change. This, in turn, may require updates to the physical configuration or DBMS configuration settings to achieve optimum performance.
- Hardware or software updates may require planned maintenance and outages of individual NoSQL nodes. Having the ability to predict performance behavior, and the ability to quickly tune the new physical configuration for optimum performance, could be a valuable tool in such situations to maintain SLAs and ensure a good user experience.

The configuration challenge is further amplified by *a shortage of experienced resources*, recently highlighted in studies by Simplilearn¹ and Deloitte². Developing a schema and tweaking the DB configuration is often handled by software developers, and is likely based on intuition or trial & error rather than expert knowledge and hands-on experience.

Not only can an optimum configuration maximize performance, but it also *reduces infrastructure and resource cost*. This, in turn, has a *positive impact on sustainability*, which has become an essential aspect of business operations, corporate culture, and corporate identity.

The main objective of this study is to evaluate the ability of Machine Learning models to make performance predictions for varying DBMS configurations *and* DB physical configurations. We

¹<https://www.simplilearn.com/why-nosql-skills-are-crucial-for-big-data-career-article> (visited on Oct. 6th, 2022)

²<https://www2.deloitte.com/us/en/insights/industry/technology/data-analytics-skills-shortage.html> (visited on Oct. 6th, 2022)

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 how the size of the training dataset influences 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 Black-box Optimization (BBO) methods that utilize the fitted ML models.

Section 2 introduces related work discovered and reviewed during the literature review that was conducted in preparation for this study. Section 3 briefly introduces the end-to-end methodology used by the authors and also defines the Tuning Domain (TD) in detail, which specifies the properties of the workload, the DBMS configuration settings, as well as aspects of the physical DB configuration that were considered within the scope of this study. Section 4 introduces the training dataset and explains the approach that was used to generate it. Results and findings are presented and evaluated in section 5. The document concludes with section 6, which discusses the results and makes suggestions for 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.

Table 1 summarizes and categorizes the related work that was reviewed as part of this study. Column *Target* specifies the type of software that the tuning approach is focused on. Some of the methods are able to tune a variety of software solutions in a generic fashion [ZLG⁺17, WLH⁺18], while others are geared specifically toward RDBMS or NoSQL technology. Column *Tuning Domain* defines what aspect of the system is considered during performance tuning. For DB solutions, the work can be categorized into approaches that tune DB performance via the DBMS configuration settings [KAS15, ZLZ⁺19, AYB⁺21, XBXY17] and solutions that tune DB performance via the physical design of the DB [CMM⁺13, BMET15, FSM⁺16], where the physical design represents aspects such as the schema design, replication, sharding, etc. Column *State* specifies *when* the tuning process takes place. *Offline* indicates that the DB is tuned in an offline state, i.e. the database is not monitored and configuration and design are not adjusted in real-time. Instead, the system is used to analyze a given configuration or make recommendations to a DBA who then actually implements the changes. *Online*, on the other hand, means that the database is monitored and configuration and design are adjusted autonomously while the system is active. *Tuning Methods* include Control Theory, Expert Systems, and a variety of Machine Learning algorithms.

One of the key differences between this work and related performance-tuning work for NoSQL

| | Target | Tuning Domain | State | Tuning Method |
|---|------------------|-------------------------------------|--------------------|---|
| <i>BestConfig</i> [ZLG ⁺ 17] | Generic | Configuration | Offline | Divide & Diverge Sampling, Recursive Bond Search |
| <i>SmartConfig</i> [WLH ⁺ 18] | Generic | Configuration | Online | Control Theory |
| <i>MAG</i> [KAS15] | RDBMS | DBMS Config. | Offline | Neural Network |
| <i>CDBTune</i> [ZLZ ⁺ 19] | RDBMS (Cloud) | DBMS Config. | Online | Deep Reinforcement Learning |
| <i>OtterTune</i> [AYB ⁺ 21] | RDBMS | DBMS Config. | Online, Offline | Gaussian Process Regression, Deep Neural Networks, Deep Deterministic Policy Gradient |
| <i>MET</i> [CMM ⁺ 13] | NoSQL | DB Design | Online | Expert System |
| <i>Bermbach et al.</i> [BMET15] | NoSQL | DB Design (Schema) | Offline | Expert System |
| <i>Farias et al.</i> [FSM ⁺ 16] | NoSQL | DB Design (Distribution) | Offline | Linear Regression, Gradient Boosting Machine |
| <i>ATH</i> [XBXY17] | NoSQL | DBMS Config. | Offline | Ensemble Algorithm and Genetic Algorithm |
| <i>PaSTA</i> [PJ20] | NoSQL | DBMS Config., DB Physical Design | Online | Adaptive Hoeffding Trees |
| <i>ConfAdvisor</i> [CHL ⁺ 21] | NoSQL | DBMS Config., OS Kernel Config. | Online | Black-box Optimization |
| <i>This Study</i> | NoSQL | DBMS Config., DB Physical Design | Offline | Ensemble Algorithm and Black-box Optimization |

Table 1: Summary of related work

systems is that the Tuning Domain includes *both* DBMS configuration settings as well as 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 Adaptive Hoeffding Tree ML model to map DB scenarios consisting of workload metrics, resource utilization, physical configuration, etc. to an ideal DBMS configuration and DB physical design. This study, on the other hand, 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. [XBXY17], except that this work utilizes Apache Cassandra instead of HBase, employs different optimization algorithms, and also includes features for the physical configuration.

3 Methodology

Apache Cassandra³ (“Cassandra”) was selected as the basis for experiments, and a prototypical implementation and domain knowledge was established, specifically focusing on the DBMS configuration settings as well as aspects of the physical configuration. A tuning domain was defined using this domain knowledge, 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 the 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 gains of the optimized configurations were evaluated through additional experiments that applied the suggested DBMS configuration and compared it against the performance of the default DBMS configuration.

Figure 1 provides an overview of the methodology and defines the terminology that is used throughout the remainder of this document:

| | | |
|--------------------|-------|--|
| x | $::=$ | A multi-dimensional input vector representing all factors that influence the performance of the DB (workload information, DBMS configuration, etc.). |
| Ω | $::=$ | The domain of x . |
| $f(x)$ | $::=$ | An unknown function that represents the actual DB performance for a given $x \in \Omega$. |
| \hat{x} | $::=$ | A multi-dimensional input vector that contains values for some of the factors that influence the performance of the database, specifically workload information, DBMS configuration settings, and aspects of the physical DB design. |
| $\hat{\Omega}$ | $::=$ | The domain of \hat{x} , the <i>tuning domain</i> (see section 3.1). |
| $\hat{f}(\hat{x})$ | $::=$ | An ML model that approximates function f to determine the DB performance for a given $\hat{x} \in \hat{\Omega}$. |

3.1 Tuning Domain

As highlighted in section 2, the majority of previous work in the field of autonomous performance tuning for NoSQL DBMSs is focused on either the DBMS configuration *or* the optimization of physical design aspects, such as the schema design or sharding strategy. Very little focus has been given to autonomous tuning of NoSQL DBs using Machine Learning with a feature set that includes both the DBMS configuration *and* aspects of the physical configuration. Both are important aspects when optimizing the performance of a database, and it may not always

³<https://cassandra.apache.org/> (visited on Dec. 22th, 2022)

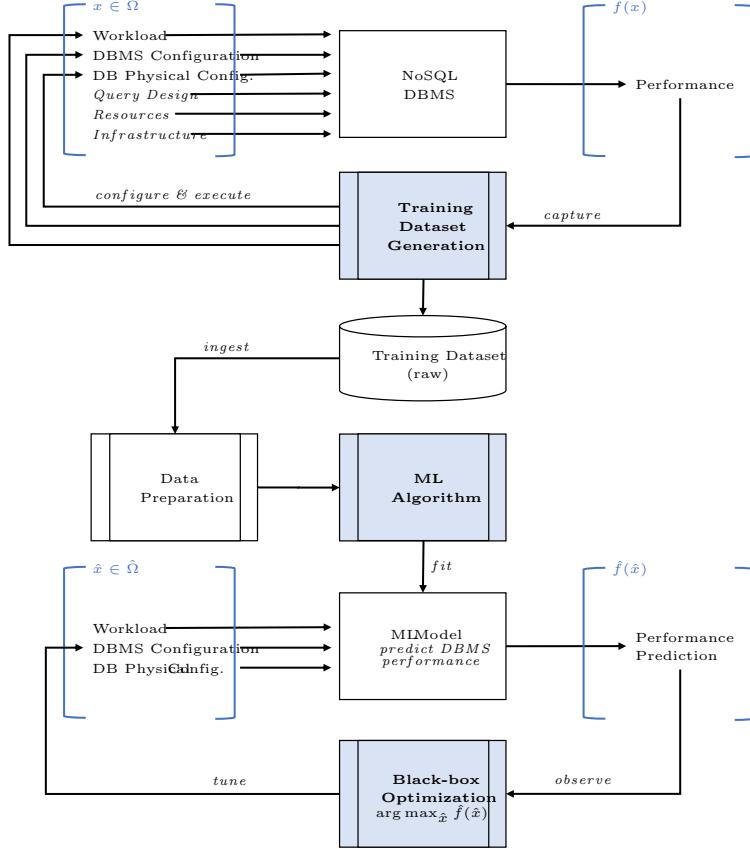


Figure 1: Overview of the methodology

be apparent whether better performance can be achieved by a different physical configuration, a different DBMS configuration, or a combination of both.

This study attempts to fit ML models to make predictions for various DBMS configuration settings and physical configurations. This section describes the Tuning Domain, i.e., the workload properties, the specific DBMS configuration settings, as well as the aspects of the Cassandra physical configuration that were considered within the scope of this study. The Tuning Domain was then used to generate a training dataset for the ML algorithms (see section 4).

Table 2 provides a summary of the features that were considered in this study along with their respective tuning domains:

Workload To fully dissect a workload and identify all aspects having an impact on performance is beyond the scope of this study. However, to follow the approach chosen in related

| 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, $2^0, \dots, 2^5\}$ |
| row_cache_size_in_mb | DBMS Configuration | {0, 20, 40, 60, ..., 200} |
| commitlog_segment_size_in_mb | DBMS Configuration | { $2^2, \dots, 2^6\}$ |
| concurrent_reads | DBMS Configuration | { $2^1 * disks, \dots, 2^5 * disks\}$ |
| concurrent_writes | DBMS Configuration | { $2^1, \dots, 2^8\}$ |
| memtable_heap_space_in_mb | DBMS Configuration | { $2^{-5} * heap, \dots, 2^{-1} * heap\}$ |
| node_count (n) | Physical Design | {2, 3, 4} |
| replication_factor (rf) | Physical Design | {1, 2, 3, 4} |

Table 2: Features included in the Tuning Domain

work, such as [CMM⁺13] and [PJ20], and to gain insight into the effect of workload composition on performance, three workloads with different distributions of read and write operations 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 Settings Apache Cassandra exposes a large amount of DBMS configuration settings to the user. The main configuration file (*cassandra.yaml*) alone contains far more than one hundred configuration settings. To reduce the complexity of this study, a subset of performance-relevant configuration settings was selected through research and targeted experiments.

Physical Design To account for aspects of the physical configuration, this study considers both *sharding* and *replication*. One of the fundamental concepts of distributed data stores is database *sharding*. Sharding is a type of partitioning, i.e., a data set is broken into smaller subsets. While partitioning does not require that the dataset be broken out across multiple nodes in a clustered environment, this fragmentation is implied by sharding [SF12]. To understand the ability of ML models to make predictions based on the sharding configuration of a database, various sharding configurations were evaluated (node count $n \in \{2,3,4\}$). A single node was purposefully omitted under the assumption that it is not a common scenario. Database *replication* within the context of this work is the practice of storing the same data in multiple locations. The replication factor (*rf*) determines on how many nodes each data record is stored.

3.2 Tuning Subdomains

One of the items that sets this study apart from the related work presented in section 2 is the inclusion of sharding and replication features. To analyze the impact of these physical configuration settings on the accuracy of the ML predictions and the models’ ability to tune the DBMS configuration, Table 3 further categorizes the Tuning Domain into TD1-TD4, which define subsets of the Tuning Domain that 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 | * |

Table 3: Definition of the tuning subdomains

TD1 represents the entire tuning domain, i.e. all of the features introduced in Table 2. This Tuning Domain is the most complex as it attempts to fit ML models that are able to make predictions for varying workloads, DBMS configurations, and physical configurations. 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, omitting both the workload and physical configuration features.

TD1-TD4 are utilized in section 5.1 to evaluate how workload and physical configuration features impact the accuracy of the fitted ML models.

3.3 Tuning Methodology

The methodology applied in this study used Machine Learning (ML), more specifically Random Forest (RF) and Gradient Boosting Decision Tree (GBDT) ML algorithms to develop models that are able to make predictions about the performance of a DB. 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.

Machine Learning (ML) Predicting performance metrics of the DB is a form of regression analysis. Whether latencies or throughput, predictions represent real numbers. *DT learning* is a powerful supervised ML methodology that can be used for both classification and regression and was chosen as the foundation for the ML objective of this study due to the strengths and benefits outlined by Géron [Gí17], including their effectiveness and unbiased nature. Decision Trees also require very little data preparation and feature scaling is typically not necessary,

and they can be used with *Ensemble Methods*. Rather than making a single model in hope of having found *the best* ML model for a particular learning objective, Ensemble Methods in ML combine the prediction of multiple ML models to make a final prediction. Two popular ensemble methods are *Bootstrap Aggregation (Bagging)* and *Boosting*. To evaluate both Bagging and Boosting methods, this study considered Random Forest (RF) and Gradient Boosting Decision Tree (GBDT) ML algorithms.

Random Forests (RFs) are a popular ensemble ML algorithm that combines multiple Decision Trees into a single predictive model. While individual Decision Trees are prone to overfitting when grown to a large depth, it has been shown using the Strong Law of Large Numbers that RFs converge with an increased number of trees in the ensemble [Bre01]. RFs use Bagging to split the training dataset into subsets that are used to train the individual DT models that are used to predict the outcome. In addition to the randomness introduced by Bagging, the RF algorithm also randomly selects the features that are considered for a split at a given node to increase the variation among the individual DTs [Bre01].

The Gradient Boosting Decision Tree (GBDT) algorithm uses a gradient-based Boosting methodology to develop an ensemble model. *Gradient Boosting* iteratively improves the ensemble model by adding ML models that are geared at minimizing the loss function using gradient descent [Fri01, NK13]. At each step in the sequence, the GBDT constructs a new decision tree attempting to offset the overall ensemble loss by correlating the new tree to the negative gradient of its loss function. Unlike RFs, the likelihood of overfitting the ensemble model increases with the number of trees in the GBDT ensemble [FCT20].

Black-box Optimization (BBO) The objective of the ML model is to accurately predict a DB performance metric for a given workload, DBMS configuration and physical design. In this sense, once fitted, it can be considered a *surrogate model* for the actual performance behavior of the database. The model itself cannot find an optimum or near optimum configuration. Instead, the ML model can be considered a black-box interface that can respond with a performance prediction for a given set of inputs. Because the model itself does not expose any meaningful information that could be used to find an optimum using a gradient-based optimization approach, this study explores BBO as a means to find an optimum configuration over the surrogate ML model.

A simple example of BBO algorithms are *Hill-Climbing algorithms*, or Local Optimization [JAMS89]. A hill-climbing algorithm is an iterative search that typically starts with a randomly or heuristically chosen value $x^{opt} \in \Omega$. It then performs a local search, i.e. slightly adjust the original value to create $x^{new} \in \Omega$. If $f(x^{new})$ is deemed better than $f(x^{opt})$, then x^{opt} is replaced with x^{new} , otherwise a different x^{new} is chosen. This process is repeated until a certain number of steps have been executed, or a sufficiently good value $f(x^{opt})$ has been found. The obvious drawback of the *hill-climbing algorithms* is that it may be focused on a *local optimum* rather than a *global optimum*.

Simulated Annealing is a popular BBO algorithm that is modeled after the physical process of annealing, which means “to subject (glass or metal) to a process of heating and slow cooling in

order to toughen, reduce brittleness, or enhance adhesion” [Har22]. The goal of the Simulated Annealing algorithm is to find an ideal or near-ideal state in a system. The algorithm is very similar to the Local Optimization algorithm except that it attempts to avoid getting stuck in a non-global optimum by adding a random element to encourage further exploration outside of the current optimum [JAMS89]. The algorithm accomplishes this by allowing an occasional move in *the wrong direction*, i.e., away from the current optimum. The likelihood that the algorithm allows this move decreases over time, just like it would in material science with a decreasing temperature.

Within the context of this study, the objective of the Simulated Annealing algorithm was to optimize the ML model’s input vector $\hat{x} \in \hat{\Omega}$ for best performance $\hat{f}(\hat{x})$ (maximum throughput or minimum latency).

4 Training Data

To train models with ML algorithms, a dataset with training examples is required. A training example contains information about the workload, the values for DBMS configuration settings, information about the physical configuration of the database, as well as information about the actual performance that was measured when the workload was executed against the DB using this particular DBMS configuration and physical configuration.

As illustrated in figure 1, the performance of a NoSQL DB depends on a variety of factors, including the hardware resources and the query design. The training dataset, which contains the actual performance measurements for a specific situation, is sensitive to these factors and therefore cannot be directly utilized for a different hardware environment or NoSQL technology. Furthermore, this study considers the number of nodes and the replication factor in the tuning domain and therefore differs from related work (e.g. [XBXY17]). Extending an existing training dataset by different physical configurations would require access to the original hardware and software environment. This was not an option, and therefore, a new dataset with the required properties was created.

| Workload: readwrite (50% read / 50% write) | | | | | | | |
|--|----|-------|--------------------|-------------------|--------------------|------|-----|
| n | rf | Count | Throughput (ops/s) | Read Latency (ms) | Write Latency (ms) | | |
| | | | Max | Min | Max | Min | Max |
| 4 | 4 | 3,068 | 77,670 | 30,962 | 10.1 | 34.8 | 3.6 |
| 4 | 3 | 3,262 | 99,393 | 24,345 | 7.4 | 28.1 | 3.1 |
| 4 | 2 | 960 | 101,398 | 44,708 | 6.8 | 20.8 | 4.2 |
| 4 | 1 | 1,048 | 164,406 | 72,613 | 4.0 | 11.1 | 3.2 |
| 3 | 3 | 1,444 | 83,787 | 30,152 | 8.3 | 34.2 | 5.3 |
| 3 | 2 | 1,267 | 80,933 | 32,536 | 8.1 | 26.8 | 5.6 |
| 3 | 1 | 1,575 | 158,199 | 66,520 | 4.0 | 12.4 | 3.0 |
| 2 | 2 | 1,566 | 71,577 | 31,835 | 11.3 | 27.9 | 5.6 |
| Workload: writeheavy (5% read / 95% write) | | | | | | | |
| n | rf | Count | Throughput (ops/s) | Read Latency (ms) | Write Latency (ms) | | |
| | | | Max | Min | Max | Min | Max |
| 4 | 4 | 1,344 | 92,760 | 61,881 | 6.4 | 15.8 | 7.0 |
| 4 | 3 | 1,433 | 111,018 | 67,962 | 4.9 | 11.3 | 5.9 |
| 4 | 2 | 1,511 | 120,197 | 78,603 | 4.5 | 8.7 | 5.6 |
| 4 | 1 | 1,221 | 153,518 | 93,332 | 4.5 | 10.0 | 4.2 |
| 3 | 3 | 1,012 | 100,949 | 57,115 | 6.4 | 14.6 | 6.6 |
| 3 | 2 | 1,251 | 110,267 | 69,294 | 4.8 | 8.7 | 5.9 |
| 3 | 1 | 1,058 | 153,563 | 91,916 | 4.1 | 7.7 | 4.3 |
| 2 | 2 | 984 | 83,530 | 51,386 | 7.1 | 11.3 | 7.8 |
| Workload: readheavy (95% read / 5% write) | | | | | | | |
| n | rf | Count | Throughput (ops/s) | Read Latency (ms) | Write Latency (ms) | | |
| | | | Max | Min | Max | Min | Max |
| 4 | 4 | 1,191 | 66,265 | 17,965 | 9.8 | 37.0 | 3.5 |
| 4 | 3 | 1,301 | 95,018 | 27,835 | 7.2 | 22.8 | 3.0 |
| 4 | 2 | 925 | 89,513 | 28,786 | 7.7 | 22.0 | 4.2 |
| 4 | 1 | 1,120 | 168,778 | 45,281 | 4.3 | 12.4 | 3.0 |
| 3 | 3 | 1,177 | 75,551 | 19,148 | 8.8 | 32.8 | 5.8 |
| 3 | 2 | 1,056 | 67,266 | 20,682 | 9.8 | 29.5 | 6.8 |
| 3 | 1 | 976 | 157,652 | 41,213 | 4.6 | 14.3 | 3.0 |
| 2 | 2 | 1,007 | 49,955 | 14,224 | 12.6 | 44.2 | 8.6 |

Table 4: Overview of the training dataset

Each training example generated used the iterative five-step methodology:

1. The Cassandra process was stopped on all nodes before generating each training example. This was necessary because some configuration settings cannot be updated while the Cassandra process is running. It also allowed for the deletion of any data that may have been cached, both on-disk and in-memory, to ensure that subsequent training examples did not benefit from unintended caching mechanisms.
2. The relevant DBMS configuration settings and physical configuration were then updated.

3. The Cassandra processes were restarted on all cluster nodes.
4. Using a workload generator, a workload was executed against the Cassandra cluster.,.
5. Relevant performance metrics were captured and stored along with metadata information that includes the values for current DBMS configuration settings, the physical configuration, and properties of the workload.

The training dataset was created on the University of Hagen’s *Newton* cluster, which, at the time of this study, consisted of five individual server nodes, newton1-newton5, each with 16 processors (Intel® Xeon® CPU E5-2630 v3 @ 2.40GHz) and 32GB of RAM. Ubuntu Linux6 version 18.04.6 LTS (Bionic Beaver) served as the Operating System (OS) on each server node. A custom process was used in combination with *cassandra-stress*⁴ to execute the workload on newton1. Apache Cassandra was installed on newton2 - newton5 to ensure that the workload process did not compete for resources with the Cassandra processes. A dataset consisting of 32,757 examples was generated, repeating the above five steps for each training example. Generating a single training example took approximately 55 seconds, totaling roughly 500 hours for the entire training dataset.

Table 4 aggregates the training examples by workload, the number of nodes (n) in the Apache Cassandra cluster, and the replication factor (rf). It lists key performance metrics captured for each group. Read and Write Latencies represent the minimum and maximum *average* latencies for each group. The performance metrics of the training dataset varied significantly. As an example, the read latency of the readwrite workload on a four node cluster (n=4) with a replication factor of 3 (rf=3) ranged from 7.4 Milliseconds (ms) to 28.1ms depending on the chosen DBMS configuration.

5 Experimental Evaluation

5.1 ML Model Quality

The quality of ML models depends on a variety of factors, including the size and quality of the dataset, feature engineering, as well as the ML methodology, training algorithm and hyperparameters chosen to fit the model. In case of supervised ML regression a variety of different quality measures exists, including the Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE) [G17].

Table 5 lists the MAE and 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.

The results highlight that the RF and GBDT algorithms produced models of similar quality. The GBDT model outperformed the RF algorithm when predicting read and write latencies. The RF algorithm, on the other hand, yielded slightly better results when predicting the overall

⁴https://cassandra.apache.org/doc/latest/cassandra/tools/cassandra_stress.html (visited on Dec. 22th, 2022)

| | 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 |

Table 5: ML Model Quality

throughput performance measure. Both ML algorithms were better at fitting models that predict latencies than they were at fitting models that predict overall throughput. This can intuitively be explained by the fact that optimizing overall throughput is an exercise that involves optimizing both read and write latencies and is, therefore, more complex.

It should be noted that the hyperparameters of the ML models referenced in this study were tuned for each ML model individually. The values for each hyperparameter varied based on the features, the tuning domain, the training dataset, and the performance metric (label) considered during a particular experiment. Hyperparameter tuning of each model had a significant impact on the accuracy of each model. For example, experiments conducted as part of this study showed that hyperparameter tuning an RF model improved the MAE measure by more than 300% compared to a model that was trained with default hyperparameter values.

The ML model quality results outlined in Table 5 were created with the entire dataset of 32,757 examples. Using a random split methodology, 75% of the dataset was used to fit the models with holdout validation (see [G17]), 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 2. 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.

To evaluate how the TD1 ML model quality compares 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 3 and confirm that the ML models trained with TD1 performed worse compared to TD2-TD4 when identically sized training datasets were used. A GBDT

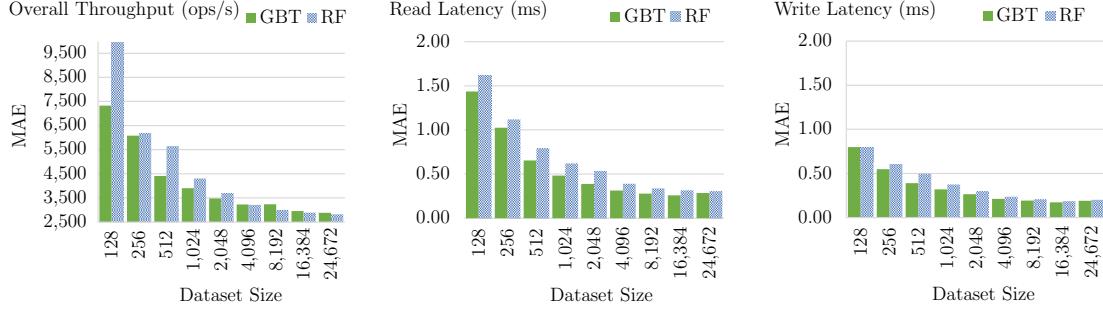


Figure 2: Influence of the dataset size on the prediction accuracy of the ML models

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.

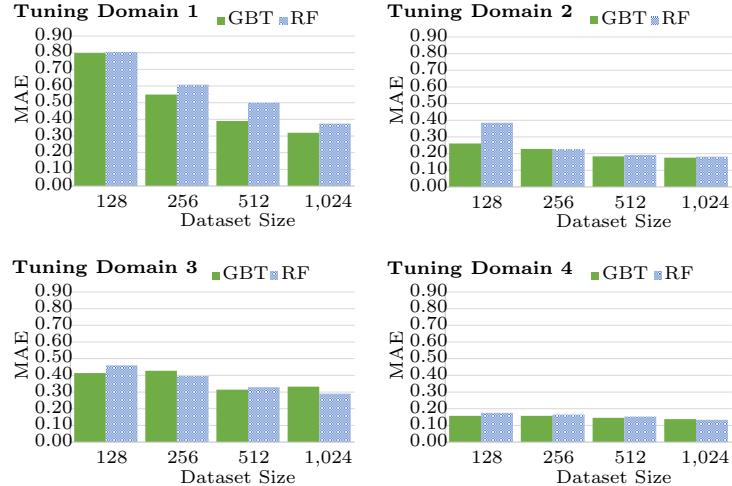


Figure 3: Influence of the dataset size on the write latency prediction accuracy of the ML models trained with different TDs

5.2 Tuning performance with Black-box Optimization

A search over the best TD1 ML model for the optimum DBMS configuration was conducted using the Simulated Annealing BBO algorithm. The search was carried out for various workloads

and a physical configuration of four nodes, and a replication factor of three. The DBMS configuration recommended by the optimization algorithm was then applied to the Cassandra cluster, respective workloads were executed, and results were captured. 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 combination of DBMS configuration and workload type, and average performance measure values were calculated. The results for all three performance measures are shown in figures 4 and 5.

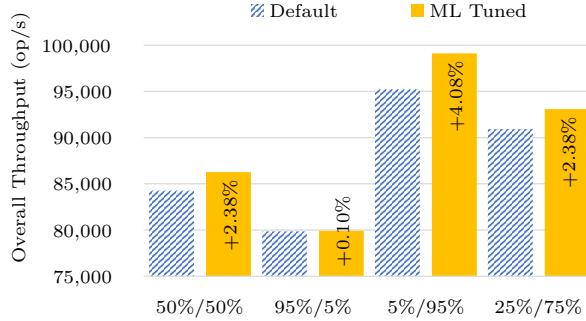


Figure 4: Actual throughput measurements for default and ML/BBO-tuned DBMS configurations

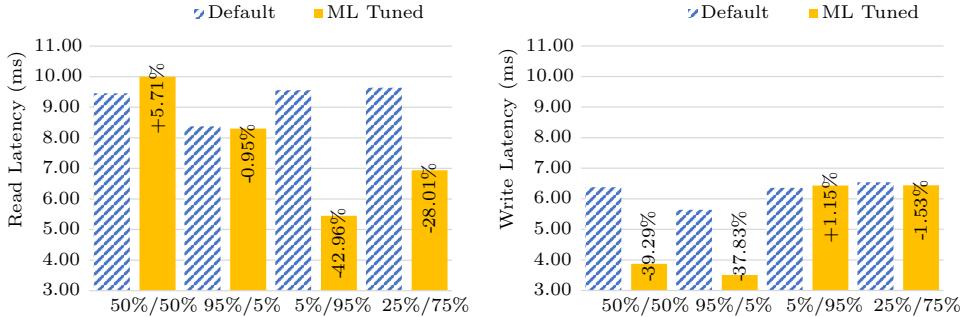


Figure 5: Actual latency measurements for default and ML/BBO-tuned DBMS configurations

Overall throughput could be improved for all workloads, most notably for the write-heavy workload (4.08%), while only a minor improvement was achieved for the read-heavy workload (0.10%). The algorithm also found a configuration that increased overall throughput for the previously unseen 25%/75% workload. More drastic improvements were achieved for the read and write latencies. Tuned configurations could reduce 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 latencies resulted in higher write latencies and vice versa, reducing overall throughput. However, these

results clearly demonstrates that the ML model was able to successfully derive knowledge which DBMS configuration settings impact the performance measures and to what extent.

Next, we applied the optimization method to different physical configurations. Similar to the previous section, a search over the best TD1 ML model was conducted using the Simulated Annealing algorithm. However, this time the objective of the algorithm was to tune the performance by adjusting the DBMS configuration under varying physical configurations. The measurements also included a physical configuration with two nodes ($n=2$) and a replication factor of one ($rf=1$) to analyze how well the methodology works for previously unseen physical configurations. While the training set included examples with two nodes and separate examples with a replication factor of one, it did not account for any examples with that particular combination.

Figure 6 shows how the throughput performance of the ML-tuned configuration compared with the default configuration (*writeheavy* workload). Similarly, figure 7 shows how the write latency performance of the ML-tuned configuration compared to the default configuration (*readwrite* workload). It should be noted that the tuned DBMS configuration settings varied across different physical configurations.

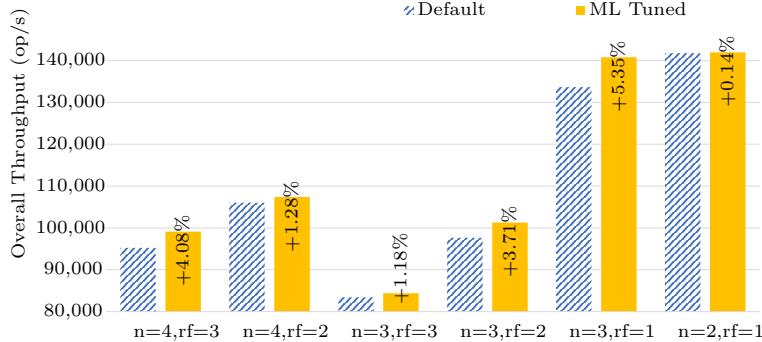


Figure 6: Actual throughput for default and ML/BBO-tuned DBMS configurations for various physical designs

These results illustrate that the ML-based tuning method could successfully tune the DB for throughput and latency under a variety of physical configurations. However, it also highlights that it failed to optimize performance for physical configurations that were not previously encountered during the training phase of the ML model. Rather than improving write latency, it increased by 12.58% from 4.57ms to 5.14ms. One reason for this may be that the ML algorithms failed to extract and derive a meaningful trend that would allow accurate predictions for physical configurations that were not included in the training dataset. Another reason could be that the corresponding features were not encoded to cultivate this kind of predictive quality in the model. Instead of using the simple ordinal encoding scheme applied as part of this study, one may consider an ordinal encoding in combination with normalization over an extended range.

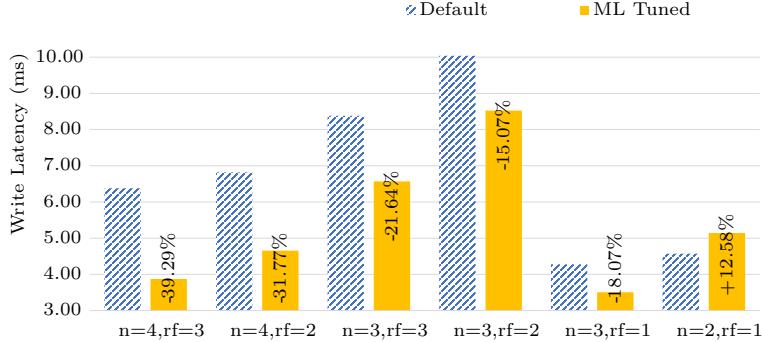


Figure 7: Write latencies for default and ML/BBO-tuned DBMS configurations under varying physical configurations

We also analyzed models that were trained with the tuning subdomains TD2-TD4. Section 5.1 showed significant accuracy differences depending on what tuning domain was used to fit the ML models, even when trained with datasets consisting of 1,048 examples. To evaluate the ability to tune the DBMS configurations with these models, the Simulated Annealing algorithm was used to search for an optimum throughput configuration for each of the TDs using the ML models fitted with 1,024 training examples. These configurations were then applied to the Cassandra cluster, the workload was executed, and performance metrics were captured. This process was repeated three times for each TD, and average throughput results were calculated. The experiments were based on the *writeheavy* workload (5% read/95% write), and results are shown in figure 8.

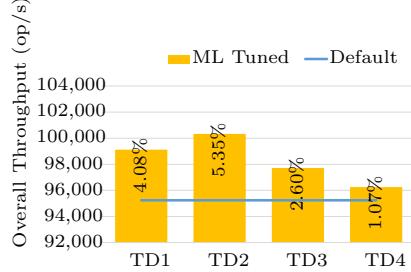


Figure 8: Actual throughput measurements for DBMS configurations that were optimized with ML models trained with TD1-TD4

Based on the observations made in section 5.1 and because TD4 clearly has the least complex feature vector, the expectation was that TD2 and TD4 would produce the most performant configurations and TD1 would produce the least performant configuration. However, this was not the case. Instead, TD1 outperformed both TD3 and TD4.

The experiment was repeated using the *readheavy* workload (95% read/5% write), this time optimizing latencies instead of throughput. The results are shown in figure 9.

These results aligned closer with expectations (TD2 and TD4 produce the best configurations).

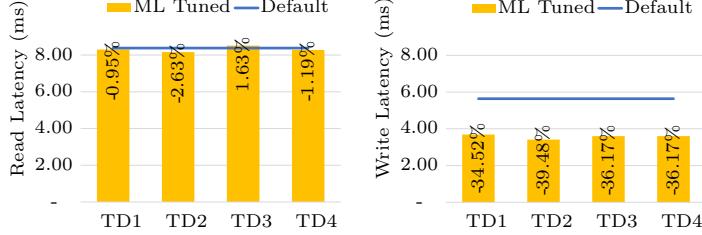


Figure 9: Actual latency measurements for DBMS configurations that were optimized with ML models trained with TD1-TD4

As illustrated in figure 3, predictions of TD2 and TD4 models were roughly twice as accurate as predictions for TD1 and TD3 models, and thus additional research was conducted to analyze why the ML models with higher accuracy were not able to produce DBMS configurations that resulted in significantly better performance. It was ultimately determined that the ability to tune the DBMS configuration depended equally on the ML model's accuracy as it did on the quality of the training data. The 1,024 training examples were chosen randomly from the training dataset. The training dataset contained a significant number of examples that resulted in performance that was worse than the default configuration. While the ML models for the less complex TDs were able to make more accurate predictions, they were not necessarily able to help the BBO algorithm find an optimum configuration if they were not fitted with any of the high-performance training examples.

6 Discussion

This study evaluated Machine Learning and Black-box Optimization as a means to performance-tune NoSQL DBs. The methodology involved fitting various RF and GBDT models using a custom training dataset to make predictions about the performance of a Cassandra NoSQL DB. When trained with the entire dataset, the ML models yielded an MAE of 3.29% for throughput and 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*. In fact, when using training datasets of 1,024 examples, the MAE of the TD1 and TD3 GBDT models were almost identical when predicting read latencies. Therefore, one may also conclude from these results 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 the Simulated Annealing optimization algorithm. The algorithm was able to find configurations for improved performance in almost all situations.

For a physical design with four NoSQL nodes and a replication factor of three, throughput improvements ranged from 0.10% to 4.08%, and 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. Experiments were also conducted involving workloads with read/write compositions that differed from the training examples. The tuned configurations improved throughput and latency performance even for these previously unseen workloads.

While the tuned DBMS configurations did result in performance improvements, none of the results matched or exceeded the performance of those training examples that exhibited the best performance. 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. The most performant training example, however, resulted in 111,018 op/s, implying an additional tuning potential of 6.5%. Some of this can likely be attributed to the fluctuations of up to 8% that were observed when measuring performance with cassandra-stress. Another potential explanation for this is the generalization capability of the ML model. While generalization helps the ML model to make more accurate predictions for previously unseen DBMS configurations, it also regulates outlier configurations. These outliers represent both subpar but 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 some of these items and highlights potential areas for future work:

- The BBO optimization methodology implemented in this study targeted individual DB performance measures (throughput *or* read latency *or* write latency). Performance objectives vary, and this may not always be desirable as performance tuning oftentimes involves finding a performance state that involves meeting multiple performance goals, e.g. reducing read latency *and* guaranteeing a certain throughput. This could be accomplished by considering multiple performance measures when optimizing the DBMS configuration. Xiong et al. approach this by combining multiple weighted performance measures into a single optimization objective [XBXY17], 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, i.e. the NoSQL database, OS, etc., that were used to generate it. This implies that the trained models are not able to make predictions for a different hardware environment or NoSQL technology. Because the generation of training examples is such a time-consuming task, it appears worthwhile to explore the feasibility of transforming or scaling training examples or derived knowledge for a different hardware environment or technology stack.
- The tuning scope of this study is limited. For example, the physical configuration is limited to the number of nodes and the replication factor. Many more aspects exist, including secondary indexes, schema design, various consistency levels, etc., that could be evaluated in more detail.
- Another observation that could be made is that the attempt 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 val-

ues. Changing the encoding scheme may improve these results.

- It should also be noted that the tuning methodology utilized in this study treats the 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. [CMM⁺13] could be evaluated as an extension to the methodology outlined in this document.
- As far as the ML approach is concerned, this study evaluated RF and GBDT to develop a surrogate model for performance predictions. Various other ML algorithms exist, and others are likely that would exhibit a similar or better prediction quality. Similarly, various other BBO algorithms exist, some of which have been shown to produce better results than the Simulated Annealing algorithms [CHL⁺21]. A more systematic evaluation of different BBO algorithms could be carried out.

References

- [AYB⁺21] Dana Van Aken, Dongsheng Yang, Sebastien Brillard, Ari Fiorino, Bohan Zhang, Christian Bilien, and Andrew Pavlo. An Inquiry into Machine Learning-based Automatic Configuration Tuning Services on Real-World Database Management Systems. *Proc. VLDB Endowment*, 14:1241–1253, 2021.
- [BMET15] David Bermbach, Steffen Müller, Jacob Eberhardt, and Stefan Tai. Informed Schema Design for Column Store-Based Database Services. In *Proc. SOCA 2015*, pages 163–172. IEEE, 10 2015.
- [Bre01] Leo Breiman. Random Forests. *Machine Learning*, 45:5–32, 01 2001.
- [CHL⁺21] Pengfei Chen, Zhaoheng Huo, Xiaoyun Li, Hui Dou, and Chu Zhu. ConfAdvisor: An Automatic Configuration Tuning Framework for NoSQL Database Benchmarking with a Black-box Approach. In *Bench 2020, Revised Selected Papers*, volume 12614, pages 106–124. Springer, 2021.
- [CL19] Jeang Kuo Chen and Wei Zhe Lee. An introduction of NoSQL databases based on their categories and application industries. *Algorithms*, 12, 5 2019.
- [CMM⁺13] Francisco Cruz, Francisco Maia, Miguel Matos, Rui Oliveira, João Paulo, José Pereira, and Ricardo Vilaça. MeT: Workload aware elasticity for NoSQL. In *Proc. EuroSys 2013*, pages 183–196, 2013.
- [FCT20] Stefanos Fafalios, Pavlos Charonyktakis, and I. Tsamardinos. Gradient Boosting Trees. *Gnosis Data Analysis PC*, 2020.
- [Fri01] Jerome H Friedman. Greedy Function Approximation: A Gradient Boosting Machine. *Source: The Annals of Statistics*, 29:1189–1232, 2001.
- [FSM⁺16] Victor A.E. Farias, Flavio R.C. Sousa, Jose G.R. Maia, Joao P.P. Gomes, and Javam C. MacHado. Machine Learning Approach for Cloud NoSQL Databases Performance Modeling. In *Proc. CCGrid 2016*, pages 617–620. IEEE, 7 2016.

- [G17] Aurlien Géron. *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media, Inc., 2017.
- [HAR16] Victor Herrero, Alberto Abelló, and Oscar Romero. NOSQL design for analytical workloads: Variability matters. In *Proc. ER 2016*, volume 9974, pages 50–64. Springer, 2016.
- [Har22] HarperCollinsPublisher. *anneal*. Houghton Mifflin Harcourt Publishing Company, 5th edition, 2022. Visited on Dec. 22th, 2022.
- [JAMS89] David S Johnson, Cecilia R Aragon, Lyle A Mcgeoch, and Catherine Schevon. Optimization By Simulated Annealing: An Experimental Evaluation; Part I, Graph Partitioning. *Oper. Res.*, 37:865–892, 1989.
- [KAS15] Abdalhamid Khattab, Alsayed Albergawy, and Amany Sarhan. MAG: A performance evaluation framework for database systems. *Knowledge-Based Systems*, 85:245–255, 9 2015.
- [LCC⁺15] João Ricardo Lourenço, Bruno Cabral, Paulo Carreiro, Marco Vieira, and Jorge Bernardino. Choosing the right NoSQL database for the job: a quality attribute evaluation. *Journal of Big Data*, 2, 12 2015.
- [NK13] Alexey Natekin and Alois Knoll. Gradient boosting machines, a tutorial. *Frontiers in Neurorobotics*, 7, 2013.
- [PJ20] Davy Preuveneers and Wouter Joosen. Automated configuration of NoSQL performance and scalability tactics for data-intensive applications. *Informatics*, 7, 8 2020.
- [QCH18] Mohiuddin Abdul Qader, Shiwen Cheng, and Vagelis Hristidis. A comparative study of secondary indexing techniques in LSM-based NoSQL databases. In *Proc. SIGMOD 2018*, pages 551–566. ACM, 5 2018.
- [SF12] Pramod J. Sadalage and Martin Fowler. *NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence*. Addison-Wesley Professional, 2012.
- [WLH⁺18] Shu Wang, Chi Li, Henry Hoffmann, Shan Lu, William Sentosa, and Achmad Imam Kistijantoro. Understanding and auto-adjusting performance-sensitive configurations. In *Proc. ASPLOS 2018*, volume 53, pages 154–168. ACM, 3 2018.
- [XBXY17] Wen Xiong, Zhengdong Bei, Chengzhong Xu, and Zhibin Yu. ATH: Auto-Tuning HBase's Configuration via Ensemble Learning. *IEEE Access*, 5:13157–13170, 6 2017.
- [ZLG⁺17] Yuqing Zhu, Jianxun Liu, Mengying Guo, Yungang Bao, Wenlong Ma, Zhuoyue Liu, Kunpeng Song, and Yingchun Yang. BestConfig: Tapping the performance potential of systems via automatic configuration tuning. In *Proc. SoCC 2017*, pages 338–350. ACM, 9 2017.

- [ZLZ⁺19] Ji Zhang, Yu Liu, Ke Zhou, Guoliang Li, Zhili Xiao, Bin Cheng, Jiashu Xing, Yangtao Wang, Tianheng Cheng, Li Liu, Minwei Ran, and Zekang Li. An End-to-End Automatic Cloud Database Tuning System Using Deep Reinforcement Learning. In *Proc. SIGMOD 2019*, pages 415–432. ACM, 6 2019.