Workload-aware Parameter Selection and Performance Prediction for In-memory Databases

Maria I. V. Lima[1], **Victor A. E. de Farias**[1],
Francisco D. B. S. Praciano[1], **Javam C. Machado**[1]

[1]Laboratório de Sistemas e Bancos de Dados (LSBD)
Computer Science Dept – UFC – CEP 60440-900 – Fortaleza – CE – Brazil

{isabel.lima,victor.farias,daniel.praciano,javam.machado}@lsbd.ufc.br

Abstract. *In-memory databases, just as hard drive ones, may offer hundreds of customizable settings, making the task of system tuning overwhelming for a database administrator. Even worse, the number of parameters continues to grow over the years and they can affect performance in a not intuitive manner. Models that capture their behavior can assist automatic tuning mechanisms to obtain optimal performance. In this work, we propose a learning-based approach to select the most meaningful parameters and generate a performance model based on both the workload and the database configurations. Experimental results confirm that our approach can create accurate performance models using only a reduced set of selected parameters.*

1. Introduction

For a long time, conventional disk resident databases were the dominant storage technology in the market. Over the years, however, the prices of physical memory have significantly decreased as shown in figure 1, while its storage capacity has increased, thus making it affordable to have whole databases fit in main memory. This is, in fact, the precise definition of an in-memory database: one where data resides permanently in memory [Garcia-Molina and Salem 1992].

The growth of in-memory databases has made it possible to achieve great performance gain, as they can reportedly be up to 50,000 times faster than disk-based systems [Lake and Crowther 2013]. This is greatly due to the fact that, comparatively, there is much less disk input/output (I/O) latency (which is known to be the biggest bottleneck in databases) being introduced in these systems. At the same time, the design of an in-memory system is very different as it has to be optimized to take advantage of memory use. Such differences may include, for instance, the absence of buffer management, use of alternative indexing structures such as T-Trees and Bw-Trees [Levandoski et al. 2013], and use of latches instead of locks. This leads to the questions of whether and how these architectural changes may reflect in the evaluation of system performance. What different aspects may become a bottleneck for such systems?

One insightful aspect to be explored regarding performance is configuration. Today's software systems are extensively customizable, often with hundreds of configuration parameters. This is true for different kinds of systems and database systems are no exception. Not only there are too many settings, but also they are constantly being renamed, excluded and added, making it difficult for a database administrator (DBA) to

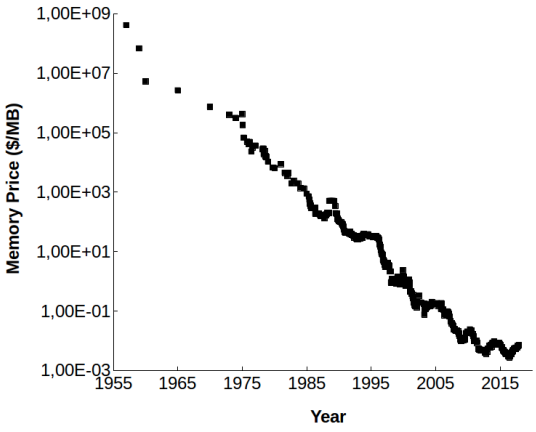

Figure 1. Cost of main memory with time [McCallum 2017]

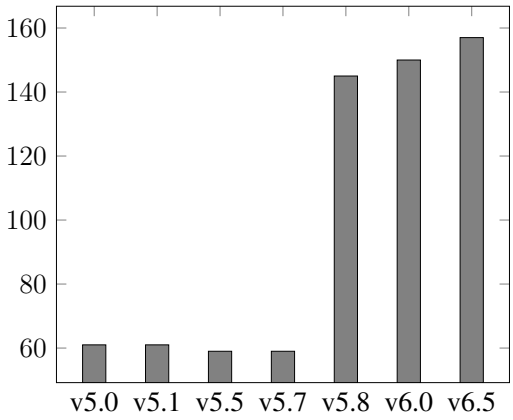

Figure 2. Number of configuration parameters in different versions of MemSQL

keep track of them all. Figure 2 shows some of these statistics for the MemSQL database system [Shamgunov 2014], where we can see how the number of configuration parameters grows over the released versions. On top of this, there are plenty of settings that have dependencies between each other, or that do not affect performance in a linear or intuitive way, or even that do not necessarily require management, i.e., they present little or no direct impact to the applications. In short, the amount of configuration parameters in modern databases makes tuning them a complex and time-consuming task.

This paper presents the results of an investigation of the impact of parameter configuration in in-memory database systems. Its contributions are:

- Identifying the minimum set of most impacting parameters, thus reducing the complexity of managing these systems;
- Analyzing the selected parameters individually and further discussing the underlying reasons for their level of importance in the context of in-memory databases;
- Providing a machine learning model for performance prediction based on the subset of selected parameters that captures the non-linear behaviour of the configuration parameters.

We model the performance of an in-memory database using two meta-parameters: workload configuration and database configuration. Workload configuration comprises a set of parameters that define a template for a workload, composed of a mix of transactions, while database configuration is defined by a collection of internal database settings, often referred to as “knobs”, provided by a manufacturer. The combination of these two meta-parameters, as it will be shown in experiments, is sufficient to provide a solid indicator of performance, whichever metric is being used to evaluate it.

2. Related Work

Several works have addressed the question of how to analyze the influence of workload and configuration parameters on the overall database performance. One way to do this is to employ statistical models so that it is possible to understand the impact of both

the configuration and the workload, thus enabling a more sophisticated analysis that can predict performance.

Ganapathi et al. (2009) proposed an approach to build a model that is capable of predicting the performance metrics accurately. To do so, the authors employ a variety of statistical machine learning techniques. In especial, they develop a model using a modified version of Kernel Canonical Correlation Analysis (KCCA). The evaluation of this model claims that it is able to predict the performance metrics in an accurate way, but it suffers from a few limitations. The main one is that the model aims to predict the performance metrics of specific queries only.

On the opposite and complementary way to the work mentioned above, Mozafari et al. (2013) explored two kinds of models, denominated black-box and white-box, that are able to predict the performance metrics. The black-box models make minimal assumptions about the context where they will be applied, whereas the white-box models make several assumptions. For the the black-box models, the authors present several machine learning regression techniques that can be used, such as KCCA and Decision Trees. In the context of white-box models, they analyze MySQL features (e.g., 2-phase locking algorithm, buffer replacement policies, etc.) to build an accurate cost-based model for the database. The results demonstrated that both of these models are able to predict the maximum throughput with relative errors within $0 - 25\%$ for a OLTP workload. It is worthy to note that none of the proposed models take into account the current settings of the database, i.e., they are solely based on SQL query logs and OS statistics that was previously collected.

Another approach is to combine different machine learning models in order to perform a feature selection before the performance analysis, because it is often hard to deal with the very large number of configuration parameters available to be adjusted in modern databases. Furthermore, evidence suggests that not all of the available settings, often in the order of hundreds, necessarily need to be tuned for a good performance [Xu et al. 2015]. A number of recent works have used this approach to develop tools and frameworks for automatically optimizing database parameters.

In order to develop a tuning tool known as iTuned, Duan et al. (2009) has proposed an approach that uses an intermediate step that is responsible for making a feature selection in order to find the highest-impact parameters. To do that, the authors employ the Statistical Approach for Ranking Database Parameters (SARD) [Debnath et al. 2008] combined with another technique denominated Adaptive Sampling. This approach is able to select the most important parameters, but one of its limitations is that it does not consider the database parameters during the feature selection process.

Likewise to the above paper, OtterTune [Aken et al. 2017] is another tool that applies feature selection as a part of a process that aims to suggests optimal configurations based on previously seen workloads and collected metrics using clusterization techniques. To do so, the authors used Factor Analysis (FA) and k-means in order to select and cluster the most important metrics. Moreover, the Lasso regression method [Tibshirani 1996] is employed to rank the knobs. Using this method, OtterTune effectively selects the proper knobs, but its focus is on the recommendation of configurations rather than on prediction.

Objectively comparing our work with all of those cited above, we observe that

none of them involve using the values of internal database parameters in the construction of the model to predict the performance indicators. Hence, the main differences from them are that i) we focus on in-memory databases, employing one in our experiments and discussing the results in light of its particularities and ii) this work aims to provide a model in which the performance is based on both the database parameters and the workload.

A comparison between the main aspects of the different works discussed in this section is shown in table 1.

Work	Database Type	Model Input	Prediction	Feature Selection
[Ganapathi et al. 2009]	Disk-based	Workload	Yes	No
[Mozafari et al. 2013]	Disk-based	Workload	Yes	No
[Duan et al. 2009]	Disk-based	Workload	No	Yes
[Aken et al. 2017]	Disk-based	Workload	No	Yes
This work	In-memory	Settings and workload	Yes	Yes

Table 1. Summary of related works

3. Our Approach

Our approach strives to select the minimum set of most impacting configuration parameters of an in-memory database while creating performance models for estimating performance metrics. We propose a data-driven machine learning-enabled strategy that employs a wrapper feature selection technique, i.e., it uses supervised learning methods for assessing the quality of subsets of configuration parameters and thus selecting the best subset.

Supervised learning is able to approximate functions by generalizing the behavior of examples. It requires a dataset composed by samples of the independent input variables (database and workload configurations) and their corresponding dependent output variable value (performance metric). Therefore, experiments are executed to generate a performance dataset to feed the learning algorithm.

In order to facilitate the understanding of the proposed solution, we divide this section in i) Dataset generation; ii) Feature selection and iii) Performance prediction.

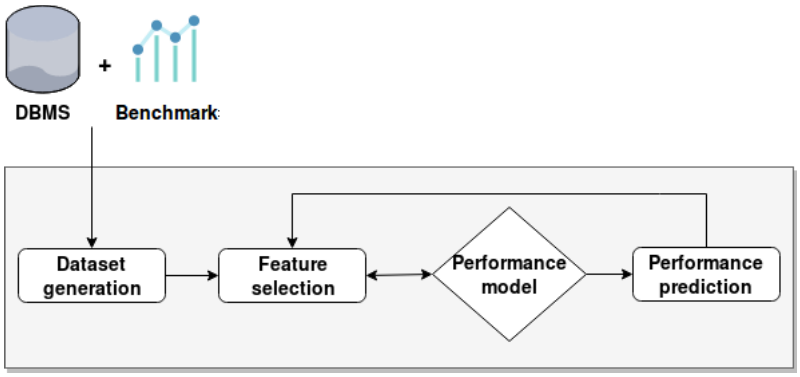

Figure 3. Approach outline

3.1. Dataset generation

Performance is dependent on both workload and database configurations (as shown in equation 1). Thus we generate a representative dataset that covers i) the domain of each configuration parameter to learn their individual influence on the performance and ii) the several combinations of variables values to expose dependencies between variables.

$$Performance = f(Config_{Database}, Config_{Workload}) \quad (1)$$

To create the performance dataset, we carry out experiments on a test environment which is a copy of the production environment. In these experiments, we vary the database and the workload configurations. The workload configuration ($Config_{Workload}$) is represented by the frequencies $< f_1, f_2, \ldots, f_t >$ where t is the number of workload templates and f_i is the frequency of the i-th workload template. For instance, in the TPC-C benchmark [The Transaction Processing Council 2007] there are five workload templates, and their combination generates different workload configurations.

The database configuration ($Config_{Database}$) comprises a group of parameters available to be adjusted by the database manufacturer. Some of these can only be set during the database creation, while others can be modified with every connection. Such details are particular to each manufacturer, with different database systems offering different sets of configuration parameters, but most of them share a great number of common parameters, even if they are not labeled the same. It is represented by $< p_1, p_2, ..., p_k >$, where k is the number of configuration parameters and p_i is the value of the i-th configuration parameter. Note that these values can be either continuous, discrete or categorical.

Thus, we execute 3,000 experiments, where each one of them corresponds to one entry in the dataset. In each experiment, random values are generated for each workload template and each database configuration parameter within their own domains, in order to produce different combinations of settings. Each experiment runs for two minutes where the first 20 seconds of warm-up and the last 20 seconds of cool-down are removed. A set of predefined performance metrics, such as latency and throughput, are collected during this process. This is represented by $< m_1, m_2, \ldots, m_j >$, where j is the number of metrics collected and m_i is the value of the i-th performance metric. In the end, the generated dataset has the following structure:

	Workload templates	Configuration parameters	Performance metrics
1	$< f_1, f_2, \ldots, f_t >$	$< p_1, p_2, \ldots, p_k >$	$< m_1, m_2, \ldots, m_j >$
...	...	...	...
n	$< f_1, f_2, \ldots, f_t >$	$< p_1, p_2, \ldots, p_k >$	$< m_1, m_2, \ldots, m_j >$

3.2. Feature selection

In this phase, we aim to eliminate the less meaningful database configuration parameters, the ones that produce little or no impact in the performance. We rely on a feature selection method that captures the most important variables based on a certain performance metric.

We employ Recursive Feature Elimination (RFE) [Guyon et al. 2002], which iteratively reduces the set of database parameters by wrapping a supervised learning method

to assess the quality of a subset of parameters. This supervised learning algorithm plays two roles in this process: i) evaluate the quality of a set of parameters by assessing their ability to predict the performance metric, as described in subsection 3.3, and ii) rank the importance of the variables, using methods like linear regression, that assigns weights to variables, or decision tree, that computes Gini coefficients for each variable.

This algorithm first considers the set of all features (parameters). In each round, it assesses the quality of this set of parameters by training a supervised model and computing its accuracy metric. Then, the trained model ranks the variables according to the weights assigned to them. As greater weights have more influence on the model, the last ranked parameter is removed and the model is re-trained using the remaining features in the next round. This process continues repeatedly until it has no remaining parameters left to test, as they have all been eliminated. The selected parameters are the ones in the round that yielded the best value for the accuracy metric.

Next, we demonstrate how the dataset referenced in section 3.1 is used in this phase. First, a performance metric m_i is chosen from the dataset as the target value, depending on the application requirements or the DBA's preference. Then, the examples (i.e. each line of the dataset) are used in the RFE algorithm to train a model using a regressor as an estimator. The attributes of the examples are the workload templates and the database configuration parameters, and the target value is the chosen metric. Each round of RFE iteratively eliminates one of these attributes.

3.3. Performance prediction

The performance prediction phase constructs models that can estimate performance metrics based on the values of the database and workload parameters. Note that the performance metrics may have a non-linear relationship with database parameters [Aken et al. 2017] and to its workload parameters [Mozafari et al. 2013]. Additionally, two parameters can show dependency [Aken et al. 2017] to each other. Predictive models should capture this behavior in order to deliver accurate estimates. This fact lead us to choose non-linear methods instead of linear models.

Non-linear models also measure the quality of a subset of features. By training a model with a given subset of variables, we can compare the prediction power of this subset to the whole set of variables. If using subset is as accurate as the whole subset, it means that the remaining variables do not account for the target value. Furthermore, in our experiments, we show that models trained with a subset of variables can be more accurate than using the whole set of variables.

As performance metrics, we use throughput and latency. Predicting these metrics is a supervised learning problem and, specifically, it is a regression task since their values are discrete and unbounded. In this work, we test several regression methods in order to evaluate which of them is more accurate for our problem.

We evaluate the quality of a model in terms of accuracy metrics. We employ the k-fold technique [Stone 1974] along with the mean absolute error (MAE) metric. k-fold computes the accuracy while being robust to both under and overfitting. k-fold divides the whole dataset equally in k folds. For each fold, it uses this fold as test dataset to compute MAE and uses the remaining $k - 1$ folds as train dataset. At the end, we take the mean value of MAE for each fold. This value is used to compare the accuracy between methods

and for evaluating subsets of parameters in the RFE algorithm. Additionally, choosing the right parameters to address can help regression methods to improve accuracy.

The generated model is able to receive a certain database and workload configuration as input and predict its performance, based on the previously seen data used to train it. Using it will save time from directly executing a workload under a given configuration to discover how long it takes to execute, for example.

4. Experimental Evaluation

In this section, we aim to evaluate our proposed approach through a series of experiments. In particular, it is our intention to show that our strategy:

- chooses the best subset of features that directly impact on performance by employing RFE, where the accuracy of predictive models is used to assess the quality of parameter subsets. The accuracy is measured using k-fold along with MAE;
- predicts performance metrics based solely on the workload and database configurations;
- provides insights for understanding the underlying mechanisms of how the configuration parameters changes the in-memory system behavior. We also provide an analysis of the main aspects behind the parameters outputted by the feature selection phase.

4.1. Experimental Setup

We relied on a commercial in-memory database that will be referred to as database A. All tests were executed in a machine with $32GB$ of RAM and 4 cores running on Ubuntu 14.04.5 LTS. For generating workloads and collecting performance metrics, we used TPC-C benchmark implemented in OLTP-Bench [Difallah et al. 2013]. While running TPC-C, a scale factor of 120 (equivalent to a database of roughly $18GB$) and a time window of two minutes of benchmark execution were used. All implementations were written in Python with intense use of the $scikit - learn$ library [Pedregosa et al. 2011] for machine learning algorithms.

4.2. Result Analysis

We tested the RFE algorithm using three different machine learning techniques: Gradient Boosting Machine (GBM) [Friedman 2001], Random Forest (RF) [Breiman 2001] and Decision Tree (DT) [Breiman et al. 1984]. In figure 4, the evolution of the applied RFE algorithm along with the number of iterations, as features are eliminated, is shown for each of the three different methods. This graphic shows that all methods eliminated the right features at the beginning of the run since MAE raises only with less than about 12 features. The difference between them is the minimum MAE and the number of features with minimum MAE.

Thus table 2 displays the minimum MAE, which is equivalent to the optimal point, obtained in each method, along with the respective number of selected features. We note that the Gradient Boosting Machine method outperformed Random Forest and Decision Tree on both MAE and the number of selected features. It selected the smaller number of features with the smaller MAE.

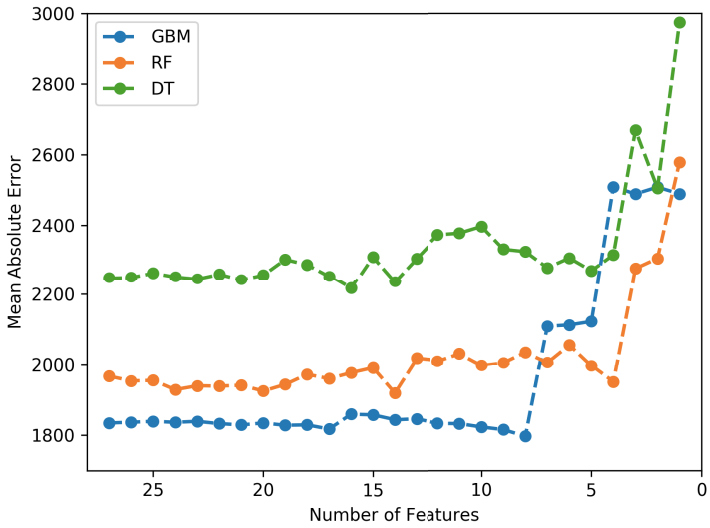

Figure 4. RFE iterations: MAE vs. Number of Features

In addition, this information is complemented by tables 3 to 5. They show, for each of the three methods, the name of the selected parameters (and their respective score) for the RFE round that yielded the best MAE. Random Forest, for example, generated a minimum MAE of 1919.7 and selected 11 parameters, which are listed in table 4 sorted by decreasing order of importance. These scores are the weights assigned to each feature by training the estimator method (GBM, RF or DT). Many parameters, like Commit Durability, Transaction Log File Size and Transaction Log Buffer Size, were selected by all methods, showing consistency between them.

	Gradient Boosting	**Random Forest**	**Decision Tree**
Min. MAE	1796.56	1919.7	2217.59
Number of Features	6	11	12

Table 2. Comparison of minimum MAE and number of selected features using different algorithms

Parameter	**Score**
Checkpoint Log Volume	0.164911
Transaction Log Buffer Size	0.147690
Transaction Log File Size	0.135332
Max. Commit Buffer Size	0.130672
Checkpoint Rate	0.128974
Commit Durability	0.119779

Table 3. Parameters selected by RFE using Gradient Boosting Machine

Next, we choose some of the selected configuration parameters shown in tables 3 to 5 to discuss about. The focus of this discussion is to better understand the importance

Parameter	Score
Commit Durability	0.212208
Lock Level	0.149018
PL/SQL Memory Size	0.062042
SQL Query Timeout	0.053479
Transaction Log Buffer Size	0.053343
Lock Wait Time	0.052239
PL/SQL Conn. Mem. Limit	0.049722
Checkpoint Log Volume	0.048706
Transaction Log File Size	0.047432
Max. Commit Buffer Size	0.046462
Checkpoint Frequency	0.042661

Table 4. Parameters selected by RFE using Random Forest

Parameter	Score
Commit Durability	0.230500
Lock Level	0.156808
SQL Query Timeout	0.058409
PL/SQL Conn. Mem. Limit	0.053595
Checkpoint Rate	0.053550
Log Buffer Parallelism	0.048701
Transaction Log File Size	0.045762
Checkpoint Log Volume	0.041598
Checkpoint Frequency	0.035915
Transaction Log Buffer Size	0.034257
Log Purge	0.029058
Max. Commit Buffer Size	0.023990

Table 5. Parameters selected by RFE using Decision Tree

of these aspects in the context of in-memory databases, bringing an insight to the reasons behind their selection.

- **Commit Durability:** Whenever a transaction is committed, its log record may or may not be immediately written to disk. Writing the transaction log to disk right after a commit means that no committed transactions will be lost in case of failure. However, this benefit comes at the expense of a lengthier execution due to the added disk access time.
- **Checkpoint Log Volume:** Checkpoints occur with a certain frequency and, between two consecutive ones, the amount of data that can be stored in the log file may be limited. If the log file is full before the next checkpoint happens, new data will not be written to the log, resulting in its loss. Thus it is important to make sure that the parameter that controls this aspect is reasonably set.
- **Transaction Log File Size:** Specifying a maximum file size for the transaction log is likely to be a bottleneck-inducing concern when tuning a database. This aspect may present an issue in the case of unknowingly having the log file being set too small, causing constant checkpoint operations (which are costly and heavily I/O-bound) every time the log file reaches its maximum size.
- **Transaction Log Buffer size:** Before being written to disk, data is generally stored in buffers in main memory. For example, it is common to have transaction commit and log buffers in database systems. Setting a buffer size too small may cause the buffer to get full very quickly, thus forcing it to recurrently flush its data to disk. Therefore, making sure that the buffers are large enough to avoid the overhead of constant disk access is an important precaution.
- **Max. Commit Buffer Size:** Similar to what happens to Transaction Log Buffer Size, limiting the size of the transaction commit buffer may cause constant disk flushing. It is good practice to make sure that the commit buffer size is large enough to avoid this happening too often.
- **Checkpoint Rate:** As checkpoint is an operation that makes use of disk access, setting the rate at which they are written to disk too low may cause the checkpoint

process to take a lot longer than expected. This may as well affect other operations, causing delays in backup and recovery times.

- **Lock Level:** This is a potentially critical parameter, as setting the lock to a not suitable level may undermine the database performance. Setting a database-level lock may significantly slow down the access of transactions. Generally speaking, row-level locking or even table-level locking tend to be better solutions to maximize concurrency control, but this is not always the case.
- **Lock Wait Time:** Deciding a suitable lock wait time acts as a double-edged sword. On the one hand, having this parameter set too high may cause transactions to wait an unnecessarily long time until they are able to obtain the lock. On the other, setting it too low will cause a great number of transactions to be aborted early, making it important to keep an eye for a balance regarding this parameter.

Ultimately, even though in-memory databases are much faster than hard disk ones due to them relying on main memory access, disk access still counts as a considerable part of the mechanics of these systems. It is possible to infer from the experiments that concurrency control also plays an important role in their performance.

The use of inadequate levels and other locking-related issues should, for sure, be carefully watched. But, in the end, disk I/O continues to be the utmost concern when looking to speed up execution times, because even in-memory databases still need some sort of persistence due to main memory's volatility. Writing and reading log files and making periodic system checkpoints are common database maintenance operations that heavily depend on I/O and inevitably introduce undesirable latency.

We note that, depending on the database used for the experiments and the configuration parameters provided, or on the generated dataset used to perform the feature selection, the results could differ accordingly. However, this approach is generic and can be applied to databases of any kind (both in-memory and disk-based). Also, it is relatively cheap to build the performance model. Once the dataset is generated (which is the most time-consuming phase), training the model can be done rather quickly.

Lastly, we assess the reduction power of feature selection. Considering that the number of original parameters in the experiments was 29, our approach achieved a reduction of 58.62% for DT, 62.06% for RF and 79.3% for GBM in the number of parameters.

5. Conclusion and Future Work

In this work, we have addressed the problem of building a performance model that takes into account both workload and database configurations. We show that it is possible to, under a given workload, reduce a potentially large set of database parameters to a smaller subset that impacts the most a chosen performance indicator. This allows database administrators to save time during the process of system tuning.

In specific, our experiments show that the use of RFE with Gradient Boosting Machine yielded the best results for the feature selection. It presented the smallest error between the three tested methods, resulting in a subset of only 6 parameters, which is a reduction of dozens of parameters.

We also analyze some of the most performance-impacting aspects regarding in-memory databases pointed out by our experiments. Upon the presented results, we con-

clude on how disk access still plays a great part in the performance of these systems, accounting for multiple possible bottleneck situations. Therefore, we highlight the importance of understanding configuration parameters and properly tuning them.

To add more supporting evidence to this work, a future extension is carrying out similar experiments to those presented here, but using a variety of different benchmarks (e.g. YCSB, TPC-H, etc.) and in-memory databases (e.g. VoltDB [Stonebraker and Weisberg 2013] or Peloton [Pavlo et al. 2017]).

Another possible future contribution is using active learning to build a precise database performance model more quickly. Instead of randomly choosing workload and database configurations in an attempt to cover the extensive search space, active learning would suggest which is the next best data point (configuration) to be labeled, thus saving time from labeling less meaningful cases.

Lastly, we plan to extend our work to automatically tune the parameters to optimize system's performance. Black-box optimization methods may be suited for this case since we do not possess the closed form of the performance function.

Acknowledgments

This research was partially funded by CAPES (grants #1697978 and #1782887) and LSBD/UFC.

References

Aken, D. V., Pavlo, A., Gordon, G. J., and Zhang, B. (2017). Automatic Database Management System Tuning Through Large-scale Machine Learning. In *Proceedings of the 2017 ACM International Conference on Management of Data, SIGMOD Conference 2017, Chicago, IL, USA, May 14-19, 2017*, pages 1009–1024.

Breiman, L. (2001). Random Forests. *Machine Learning*, 45(1):5–32.

Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, C. J. (1984). *Classification and Regression Trees*. Wadsworth.

Debnath, B. K., Lilja, D. J., and Mokbel, M. F. (2008). SARD: A statistical approach for ranking database tuning parameters. In *Proceedings of the 24th International Conference on Data Engineering Workshops, ICDE 2008, April 7-12, 2008, Cancún, México*, pages 11–18.

Difallah, D. E., Pavlo, A., Curino, C., and Cudré-Mauroux, P. (2013). OLTP-Bench: An Extensible Testbed for Benchmarking Relational Databases. *PVLDB*, 7(4):277–288.

Duan, S., Thummala, V., and Babu, S. (2009). Tuning Database Configuration Parameters with iTuned. *PVLDB*, 2(1):1246–1257.

Friedman, J. H. (2001). Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232.

Ganapathi, A., Kuno, H. A., Dayal, U., Wiener, J. L., Fox, A., Jordan, M. I., and Patterson, D. A. (2009). Predicting Multiple Metrics for Queries: Better Decisions Enabled by Machine Learning. In *Proceedings of the 25th International Conference on Data Engineering, ICDE 2009, March 29 2009 - April 2 2009, Shanghai, China*, pages 592–603.

Garcia-Molina, H. and Salem, K. (1992). Main Memory Database Systems: An Overview. *IEEE Trans. Knowl. Data Eng.*, 4(6):509–516.

Guyon, I., Weston, J., Barnhill, S., and Vapnik, V. (2002). Gene selection for cancer classification using support vector machines. *Machine Learning*, 46(1-3):389–422.

Lake, P. and Crowther, P. (2013). In-memory databases. In *Concise Guide to Databases*, pages 183–197. Springer.

Levandoski, J. J., Lomet, D. B., and Sengupta, S. (2013). The Bw-Tree: A B-tree for new hardware platforms. In *29th IEEE International Conference on Data Engineering, ICDE 2013, Brisbane, Australia, April 8-12, 2013*, pages 302–313.

McCallum, J. C. (2017). Memory prices (1957-2017). `https://jcmit.net/memoryprice.htm`. Accessed: 2018-03-05.

Mozafari, B., Curino, C., Jindal, A., and Madden, S. (2013). Performance and resource modeling in highly-concurrent OLTP workloads. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2013, New York, NY, USA, June 22-27, 2013*, pages 301–312.

Pavlo, A., Angulo, G., Arulraj, J., Lin, H., Lin, J., Ma, L., Menon, P., Mowry, T. C., Perron, M., Quah, I., Santurkar, S., Tomasic, A., Toor, S., Aken, D. V., Wang, Z., Wu, Y., Xian, R., and Zhang, T. (2017). Self-Driving Database Management Systems. In *CIDR 2017, 8th Biennial Conference on Innovative Data Systems Research, Chaminade, CA, USA, January 8-11, 2017, Online Proceedings*.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., VanderPlas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Shamgunov, N. (2014). The MemSQL In-Memory Database System. In *Proceedings of the 2nd International Workshop on In Memory Data Management and Analytics, IMDM 2014, Hangzhou, China, September 1, 2014.*

Stone, M. (1974). Cross-validatory choice and assessment of statistical predictions. *Journal of the royal statistical society. Series B (Methodological)*, pages 111–147.

Stonebraker, M. and Weisberg, A. (2013). The VoltDB Main Memory DBMS. *IEEE Data Eng. Bull.*, 36(2):21–27.

The Transaction Processing Council (2007). TPC-C Benchmark (Revision 5.11). `http://www.tpc.org/TPC_Documents_Current_Versions/pdf/tpc-c_v5.11.0.pdf`.

Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 267–288.

Xu, T., Jin, L., Fan, X., Zhou, Y., Pasupathy, S., and Talwadker, R. (2015). Hey, you have given me too many knobs!: understanding and dealing with over-designed configuration in system software. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2015, Bergamo, Italy, August 30 - September 4, 2015*, pages 307–319.