Too Many Knobs to Tune?
Towards Faster Database Tuning by Pre-selecting Important Knobs

Konstantinos Kanellis, Ramnatthan Alagappan, and Shivaram Venkataraman

University of Wisconsin – Madison

Abstract. To achieve high performance, recent research has shown that it is important to automatically tune the configuration knobs present in database systems. However, as database systems usually have 100s of knobs, auto-tuning frameworks spend a significant amount of time exploring the large configuration space and need to repeat this as workloads change. Given this challenge, we ask a more fundamental question of how many knobs do we need to tune in order to achieve good performance. Surprisingly, we find that with YCSB workload-A on Cassandra, tuning just five knobs can achieve 99% of the performance achieved by the best configuration that is obtained by tuning many knobs. We also show that our results hold across workloads and applies to other systems like PostgreSQL, motivating the need for tools that can automatically filter out the knobs that need to be tuned. Based on our results, we propose an initial design for accelerating auto-tuners and detail some future research directions.

1 Introduction

Tuning configuration parameters is critical for achieving high performance in database systems. This has been true in the past [8, 12, 20] and recent research shows that this is still the case [1, 23, 24]. However, database systems typically have hundreds of configuration knobs that determine the system's runtime behavior [22]; for example, PostgreSQL has about 170 knobs [23] and Apache Cassandra has around 155.

Manually finding the important knobs and their optimal values is thus a daunting task even for experienced practitioners. Consequently, researchers have resorted to automatic ways to tune knobs [9]. Recent auto-tuning methods for database systems typically fall into two classes: search-based approaches [24] and learning-based (ML) methods [1, 23]. In both cases, one starts with a set of knobs and a range of valid values for them, and then the tuning system tries to find the optimal values for each knob to maximize or minimize a specified objective (e.g., throughput or latency). The time it takes to tune is directly dependent on the number of knobs that needs to be tuned [23, 24]. With a larger number of knobs, the search space for search-based methods expands vastly, making it slower to find optimal values for all knobs. Similarly, ML-based approaches require a larger training dataset as we increase the number of knobs, with representative samples required from different values for each knob.

Given this, in this work, we ask the following fundamental question: *how many knobs do we need to tune to achieve good performance? Is this a smaller subset of the entire set?* To answer this question, we conduct a detailed and systematic measurement-based study. Specifically, we measure the performance of a target system under different configurations and use learning-based techniques to find how many and which knobs need to be tuned for high performance. Our results surprisingly show that tuning just *five* knobs can achieve 99% of the performance achieved by the best configuration (which tunes many knobs) for the YCSB-A workload in Cassandra.

Motivated by this result, we also extend our experiments across different workloads and systems. Our experiments show that similar results do indeed hold for different workloads (e.g., YCSB-B in Cassandra) and across different systems (PostgreSQL running YCSB-A and YCSB-B). Finally, we also compute the similarity between the sets of important knobs across workloads and find that there is a significant overlap of important knobs across some workloads.

Based on the above results, we argue that it is possible to significantly accelerate auto-tuning approaches by automatically filtering the most important configuration knobs. We propose a new design where we run a filtering or pre-selection phase to determine the important knobs, and we present some initial approaches to realize this design. We also describe future directions for research in understanding how different hardware or metrics change important knobs, and how we can perform auto-tuning while maintaining reliability guarantees.

2 Background and Motivation

We begin by providing a brief background on auto-tuners and describing the challenges in existing tuning frameworks.

2.1 Automatically Tuning Database Systems

Exploring the entire space of configuration knobs in database systems is time-consuming and intractable. Three factors contribute to this problem. First, modern systems have too many knobs. Second, most knobs take values from a large continuous space, leading to many possible configurations. Finally, it may take several minutes to measure how even a single configuration performs [13, 24].

As a result, recent research has proposed auto-tuning tools for database systems (e.g., Ottertune [1], BestConfig [24], and CDBTune [23]). Most of these tools rely upon an initial offline phase, which either builds a knowledge base that is then used to bootstrap the tuning process for future workloads (by reusing already evaluated configurations), or trains an ML

model that is used to recommend configurations during the online tuning phase. Performing this initial profiling phase is vital to the quality of configurations that these tools produce. For instance, the authors of CDBTune [23] report that their system performs better compared to search-based systems like BestConfig, as the latter cannot find optimal configurations in a short time without prior knowledge.

2.2 Challenges in Auto-tuning

The offline profiling (or training) phase, while crucial for obtaining high-quality configurations, is unfortunately very expensive. In fact, it is the most time-consuming step in the entire pipeline of these tools, accounting for over 95% of the total tuning time (i.e. several hours to days) [1, 23]. In addition, this phase requires substantial hardware resources in order to be executed in a reasonable time (multiple machines for parallel evaluation of different configurations). Further, this phase typically has to be re-executed for new workloads, or when porting the database system to a different hardware. Overall, the long running time of the offline phase and that it has to be run many times is one of the key challenges in using existing database auto-tuning tools in the real-world.

One approach to address this issue is to reduce the number of knobs that the auto-tuner must tune, thus reducing the search space or the training dataset size, which in turn cuts down the overall tuning time. However, the reduced set must contain the most important knobs, i.e., knobs that have the most effect on the performance; tuning irrelevant knobs will not lead to high performance. Most prior frameworks, however, ignore the importance of the knobs [13, 23, 24]. Some tools (e.g., OtterTune) do have an additional step in the pipeline to filter out knobs that are redundant or do not impact performance much [1]. However, such tools do the filtering step only *after* performing the initial profiling phase and observing the performance of the proposed configurations.

Given this, our primary goal in the paper is to study if it is possible to reduce the number of knobs that auto-tuners consider, while still yielding high performance. To realize this goal, a few key questions must be answered. First, how many and which knobs does one need to tune to obtain high performance for a given workload? Second, does this set of knobs change across workloads and hardware?

Recent research concurrent to our work has examined some of these questions in the context of file systems [6]. Our work focuses instead on database systems, which usually have a more complex configuration space than file systems. This complexity is reflected in the number and the nature of database configuration knobs. Database systems have ~4× more performance-related knobs compared to file systems [6] and most knobs in databases take continuous values, while most knobs are discrete in file systems. Therefore, while it is possible to evaluate all configuration points for a specific file system, this is impossible for a database [6]. Further, typically no common knobs exist among different database systems (in contrast, for example, knobs like block-size and inode-size are common to many file systems) [7]. We next discuss our study of important knobs for database systems and also compare our findings to observations made by prior work.

3 Study Methodology

Our study methodology consists of two main phases. In the first phase, we collect information on what performance numerous points in the configuration space deliver. In the second phase, we apply machine-learning methods to the collected data points to determine the most important knobs.

3.1 Generating and Collecting the Samples

Given that the configuration space is vast, it is prohibitively time-consuming to collect performance measurements for every single point in the space. We thus use a sampling-based approach. We employ a stratified sampling method called Latin Hypercube Sampling (LHS) [18], a well-known sampling technique used by many systems [1, 6, 13].

LHS takes as input the number of samples (N) to be generated. For each knob, LHS splits its value range into N equal-sized intervals. Then, N samples are selected from this space, such that each interval of any knob contains *exactly one* sample in it. Thus, LHS is able to generate samples that thoroughly and uniformly cover the configuration space. Prior work has shown that LHS covers the space more effectively compared to alternative techniques such as random or Monte Carlo sampling for a given target number of samples [10, 18].

Each sample generated by LHS corresponds to a single point in the entire configuration space. For each such point, we arrange the system under test (e.g., Cassandra) to use that point as its current configuration. Then, we run a target workload (e.g., YCSB-A) and record the performance metrics (e.g., throughput or average latency) for that point. The output of the first phase is thus a dataset that consists of points in the configuration space and the obtained performance metrics.

3.2 Quantifying Knob Importance

Our goal is to determine which knobs have the most effect on performance, i.e., to find which knobs and the target metrics (e.g., throughput, average latency) have a strong correlation. Since our target metrics are continuous, we approach this problem using regression.

Regression models estimate the value of a dependent variable (y), given a set of independent variables (X) [15]. In our case, the independent variables correspond to the database-system knobs and their values, and the dependent variable(s) to the target metric(s). For a regression model, the goodness of its fit can be measured using the *coefficient of determination*, R^2. R^2 indicates the proportion of the variance in the dependent variable that can be explained by the independent variable(s) [16]. Higher values correspond to better fits; a value of 1 implies a perfect fit. We use R^2 values to evaluate

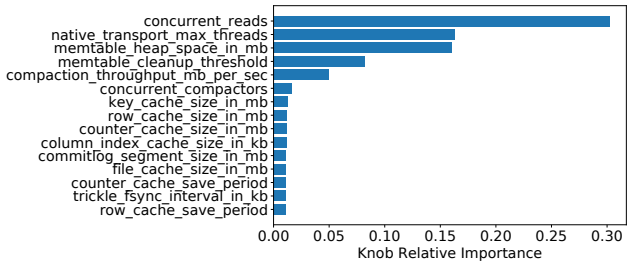

Figure 1: **YCSB-A**

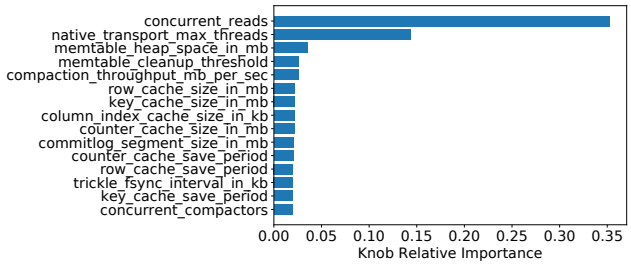

Figure 2: **YCSB-B**

Most important Cassandra knobs for YCSB-A and YCSB-B for throughput from 25K samples (top 15 knobs).

how well a model that is fitted using a smaller subset of knobs performs, compared to the one that is fitted on all knobs.

In order to quantify the importance of each knob, we use an ensemble of regression trees (i.e., random forest). Tree ensembles are generally more resistant to noise and overfit less than a single tree [4]. Regression trees are non-parametric regression models that utilize variance-based metrics to select which knobs will be included in each tree node. We use the classification and regression trees (CART) [5] algorithm, which quantifies the importance of a knob based on the reduction in variance of the target metric (dependent variable). CART builds the tree by greedily selecting the "best" knob (i.e., one that reduces variance significantly) for each node in a top-down fashion. Since our random forest consists of many trees where each tree is fitted on a random subset of samples, the "importance score" is averaged across all trees.

Random forests are an appropriate choice for our approach as they capture non-linear relationship among variables, are interpretable, and can be trained fast. Other popular alternatives, for example, linear regression, can provide interpretability but fall short on capturing non-linear relationships. More sophisticated models, e.g., neural networks, can capture complex relationships, but are expensive to train and usually are a black-box thus undermining interpretability. Due to their hierarchical structure, random forests are also able to capture variable (knob) interaction effects [21]. Therefore, we do not need to explicitly include interaction terms in the set of independent variables, as with other regression methods (e.g. linear regression with polynomials). However, while it is not easy to identify or examine the importance of a specific interaction between two (or more) knobs, the interaction factors are included in the importance score of the individual knobs. Thus, if a strong interaction exists between two knobs, then the model will give high importance score to each of them. If they eventually get included in the set of important knobs, their interaction will be accounted for during the tuning phase.

Before fitting the data we perform two pre-processing steps. We first transform all categorical knobs into dummy variables. Since the number of categorical knobs is small (2 each for the two database systems we analyzed), the number of knobs remains manageable. Second, we standardize the knob values (i.e., scale to zero mean, unit variance). This ensures that our regression model is not influenced by the different range of values of each individual knob. At the end of the second phase, using the regression models, we can arrive at the set of important knobs and quantify their relative importance.

4 Analysis

We now present our analysis to answers the questions posed. We start by describing our experimental setup.

4.1 Experimental Setup

Target Systems and Workloads. We study Cassandra (v3.11) [2], a NoSQL store, using the YCSB-A workload from the YCSB suite [11]. This workload is write-heavy with 50% reads and 50% writes. We first determine the important knobs for this workload (§4.2). Then, to study if the important knobs change with workloads, we study Cassandra using YCSB-B, a read-heavy workload with 95% reads and 5% writes (§4.3). Finally, to see if the results hold across systems, we study PostgreSQL v9.6, a relational database, using the same workloads (§4.4).

Sample Generation and Collection. The total number of knobs for Cassandra is 155, and for Postgres 169. We generate 25K samples in the configuration space per system. These samples are obtained by tweaking 30 performance-related knobs for Cassandra, and 29 for PostgreSQL. We hand-picked these knobs by consulting the documentation and prior works [1, 23]. Each of the authors independently examined all knobs and created a set of knobs that they believed would have some measurable impact on system performance. The final set of knobs is the union of the three individual sets. It is worth noting that tweaking all knobs would have led to much smaller coverage of the configuration space, due to relative small number of samples generated by LHS compared to the entire configuration space. This might have led to insufficient data for our machine-learning model. In general, we do not expect large deviations for the top-10 (or top-20) knobs, when considering all knobs. Before running the workload, we initialize the system under test with a single table with 18 million tuples. We then run the workload with 50 client threads. For each experiment (configuration), we run the workload for five minutes and record the overall throughput and the read- and write-latency statistics at the end. Our client-threads setting is similar to those used in prior work [1, 23].

Best configuration (samples, knobs)	Apache Cassandra			PostgreSQL		
	Throughput (ops/sec)	**Read latency (usecs)**	**Write latency (usecs)**	**Throughput (ops/sec)**	**Read latency (usecs)**	**Write latency (usecs)**
Baseline (25K, 30)	74780.33	744.34	302.82	14134.34	907.37	4219.44
Validation (4K, Top-5)	74304.42	750.56	307.08	14006.90	967.52	4238.93
% of Baseline	**99.36%**	**100.84%**	**101.41%**	**99.10%**	**106.63%**	**100.46%**

Table 1: Performance when evaluating database with fewer samples that modify fewer important knobs, for workload YCSB-A.

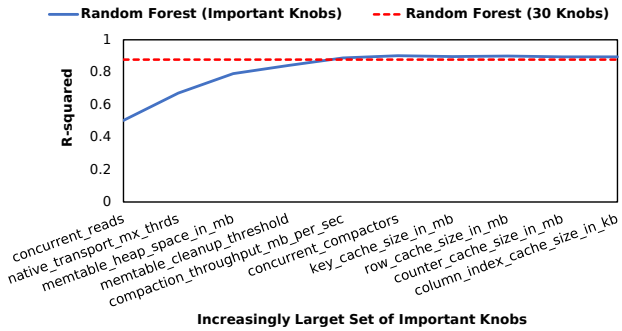

Figure 3: Baseline model performance (30 knobs – red dotted line) vs. increasingly larger sets of top knobs (blue line).

We parallelize sample collection using 30 machines, with each machine running one experiment at a time. All machines have identical hardware specifications, run Linux v4.15, and are a part of the CloudLab infrastructure [14]. We run the database system and the YCSB clients on the same physical machine but isolate them on separate CPUs (each machine has two CPUs). On each machine, the database system runs on a 10-core Intel Xeon Silver 4114 CPU with 64 GB of memory and uses a 480-GB SSD for storage. Running each experiment takes $\sim$ 9 minutes, and thus collecting 25K samples for a single system-workload pair requires approximately 3750 node-hours (or a bit over 5 days when using 30 nodes).

Analysis. We use the Python's scikit-learn implementation of random forest (RF) for our regression model, which uses the CART algorithm. We initialize each RF with 300 trees, which lies inside the range of values (i.e. $[128, 512]$) that is shown to provide a good trade-off between model performance and training time [19]. Training the model on the collected 25K samples takes under a minute on a single machine.

4.2 How Many Knobs Matter?

We analyze the samples to find the important knobs for obtaining high throughput for YCSB-A in Cassandra. Figure 1 shows the ranking of knobs and their relative importance. The key result is that the top *five* knobs are much more important than the other ones. This importance reflects the amount of reduction in variance of our target metric (i.e., throughput).

From our regression model, the most important knob is `concurrent_reads`, which determines the number of simultaneous read operations that can be performed. We believe one reason this knob ranks high is because reads are more expensive than writes in Cassandra since most reads need to go to disk [3]. Thus, increasing this knob's value allows many reads to be concurrently issued, allowing the drive to batch and (if necessary) reorder the requests, leading to higher throughput. The next important knob is `native_transport_max_threads` which is the maximum number of threads used to handle requests; with higher values, many threads can handle many concurrent client requests, improving throughput. The next three knobs determine the amount of memory allocated to the memtable, what percentage of this space can be filled before flushing to disk, and the rate at which new SSTables are written. More analysis is needed to explain why these knobs are ranked high.

Next, we use this ranking to fit our regression model with increasingly larger sets of the most important knobs (Figure 3). The baseline model is fitted on all 30 knobs, and can explain $\sim$ 90% of the variance (the red dotted line). We observe that a model fitted with just the most important knob is able to capture $\sim$ 50% of the throughput variance. Including the second knob raises this value to $\sim$ 65%. We continue to see improvements for the first five knobs, after which we see little or no improvements. The slight performance improvement of models fitted with fewer knobs over the baseline is because of overfitting, which occurs due to the large number of input features (knobs). Thus, we hypothesize that tuning as few as *five* knobs might be enough to reach almost the same performance obtained by the best among the 25K configurations.

In order to validate this hypothesis, we generate a separate set of 4K samples but by modifying only the values of the five most important knobs; the remaining ones are assigned their default values. We measure the performance of these configurations for YCSB-A and Table 1 shows the results for three target metrics: throughput, average read latency, and average write latency. We observe that just by tuning the five most important knobs, we are able to reach almost the same level of performance (for all three metrics) obtained by the best configuration in the 25K samples. Prior work [6] has also done analysis that suggests that only a few knobs are important for performance in file systems. We find this also to be true for databases and we also use experiments to validate that just tuning five important knobs can be sufficient.

Summary and Implications. Tuning just a few most important knobs can yield high performance. Based on this result, we believe the tuning time of existing auto-tuners can be significantly reduced by pre-filtering such important knobs.

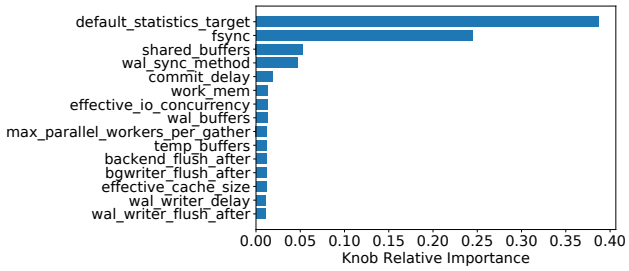

Figure 4: **YCSB-A**

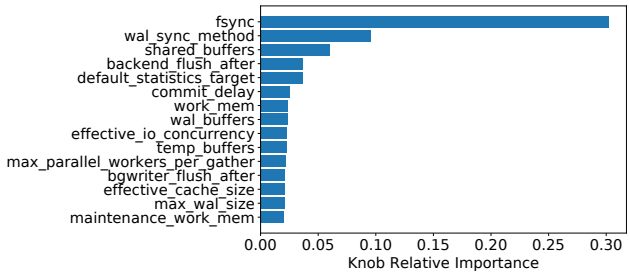

Figure 5: **YCSB-B**

Most important PostgreSQL knobs for YCSB-A and YCSB-B for throughput from 25K samples (top 15 knobs).

4.3 Do Similar Results Hold for a Different Workload?

We next conduct a similar analysis for YCSB-B and Figure 2 shows the most important knobs that impact throughput in Cassandra. We observe that a few knobs (~ 5) are more important than others, providing more evidence to our hypothesis that only a few knobs affect performance the most. We observe that `concurrent_reads` is ranked again as the most important knob, and with a greater score (than in YCSB-A); this is because YCSB-B is a read-heavy workload and thus increasing the concurrency in reads has even more effect.

More interestingly, we see that the five most important knobs exactly match the ones for YCSB-A. We believe the reason for this is that the top five knobs for YCSB-A were all important for improving read performance (writes are anyway cheaper than reads in Cassandra [3]); as a result, the model ranks the same knobs high for a read-heavy workload.

From the above preliminary observation, we believe that it might be possible (at least across some workloads) to find the important knobs once for one workload and tune the same knobs to obtain high performance for a different workload. While this seems to be true for improving throughput in Cassandra across YCSB-A and YCSB-B, we anticipate that we may not get such an exact overlap across very different workloads (read-only vs. write-only) or if we consider a different metric (e.g., write latency.) In such cases, although there is no exact overlap, we think it might be possible to take a larger set of knobs from one workload (say 15) and analyze if tuning them alone leads to high performance for a different workload. We believe this is an interesting avenue for future work.

Summary and Implications. Our hypothesis that only a few knobs impact performance significantly seems to hold for a different workload. We also note that across some workloads, there is a significant overlap of important knobs. Based on this, we believe it might be possible to find the important knobs once and tune the same or a slightly larger set of knobs to achieve high performance for a different workload.

4.4 Do Results Hold across Database Systems?

We now turn our focus to PostgreSQL to check if our findings hold true for a different system. For YCSB-A (Figure 4), similar to Cassandra, we observe that a handful of knobs are most important. Among them, `default_statistics_target` is the most important one; we observe that higher values for this knob collects highly accurate table statistics for query optimization but hurts performance; lower values lead to higher performance. `fsync` enables or disables forced writes to storage; given that 50% operations are writes in YCSB-A, this knob has a large effect and thus is ranked high.

We also ran a similar experiment as in §4.2 to find how close can we get to the performance achieved by the best sample. We generate 4K samples by modifying only the five most important knobs and measure the performance (Table 1). We again notice the we can closely approximate the performance of the best configuration, by only modifying five knobs.

We also run PostgreSQL with YCSB-B and plot the most important knobs in Figure 5. Even though YCSB-B has 95% reads, we find that the most important knob is `fsync` and the second most important knob is `wal_sync_method`. Looking more closely at our data, we find that configurations with `fsync` off typically have around 30%-80% higher throughput than those with `fsync` on. We believe that this behavior happens because PostgreSQL tries to serialize read and write operations to the same key and YCSB-B uses a Zipfian key distribution. We hope to perform additional investigation in the future to isolate and explain this behavior.

Summary and Implications. Our observation that only tuning a handful of knobs is sufficient generalizes to other systems like PostgreSQL across multiple workloads.

5 Towards Faster Database Tuning

Our preliminary study shows that only a few knobs have the most effect on performance. Thus, we argue that the tuning time of existing tuners can be significantly reduced if important knobs are identified before running the tuner. We propose a two-level design where we first run a *pre-selection* step that only identifies the important knobs (but not their optimal values). After that, we reuse existing tuners to determine the optimal values for the knobs. Our key insight is that filtering cuts down the search space or the training dataset of existing tuners, thus reducing the tuning time. For example, we consider an existing tuner, BestConfig with Cassandra on YCSB-A. We observe that when tuning only the top-5 knobs, BestConfig manages to reach the best performance using $5\times$ fewer iterations, compared to when tuning all 30 knobs.

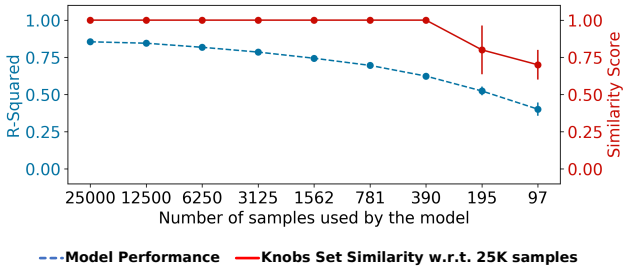

Figure 6: Model performance and top-5 knobs similarity when model is fitted with less samples (Cassandra YCSB-A).

However, as we described in §4, our current analysis uses ~25K samples. While finding the important knobs from such a large data set makes our analysis robust, it comes at a cost: the time it takes to collect the samples. Each sample in our dataset is run with a specific configuration and workload for five minutes. Thus, collecting the samples is time-consuming, even with several machines running in parallel. Using the same method to optimize existing tuners would not result in dramatic improvements in tuning time.

One way to expedite this process is to examine if we can arrive at the same results (i.e., the same knobs) with fewer samples. We conducted an experiment to study this question and our initial results are shown in Figure 6. We compare the similarity score (intersection-over-union index [17]) of the top-5 important knobs and R^2 obtained when using random subsets of samples against that of the baseline (25K samples). As shown, although the R^2 scores go down as we reduce the sample-set size, even with 64× fewer samples, we obtain the same important knobs (i.e., a similarity score of 1). Therefore, instead of using 3750 node-hours (25K samples) to collect samples, we would use $\sim$ 60 node-hours (400 samples). Assuming that the set of top-5 (or top-10) knobs remains the same across workloads, this could also reduce the need to execute experiments with many workloads beforehand (as it is the case with existing tuners [1, 23]). We believe more analysis is needed to determine how far we can reduce the sample-set size while being able to find the important knobs for different workloads and systems.

Another question that needs to be addressed is *how many* knobs should the pre-selection step output. One way to pick this would be to see how many knobs are required to get a good fit i.e., a target R^2 score. Another way is to consider the time budget for tuning. Given a time budget, there are many ways to split it between the two phases: a) identifying important knobs, and b) determining the optimal values for these knobs. If the pre-selection step outputs more knobs, it is inevitable that more time is needed by the tuner for exploration to come up with an optimal configuration. Hence, if we can only run the tuner for a short duration, then it is imperative that the pre-selection step produces fewer knobs.

A completely different approach is to use white-box knowledge of the target system to determine which knobs could have the greatest impact. We plan to investigate a number of avenues including techniques to parse comments or using profiling tools to detect which knob affects performance the most. We believe that latent information present in the code can be useful for selecting important knobs. We believe investigating these approaches can make the pre-selection step faster while finding configurations that provide high performance.

6 Conclusion

In this paper we studied the question of how many knobs do we need to tune to achieve good performance in database systems. Our results indicate that tuning a handful of knobs is good enough and that this trend holds across workloads and database systems. Based on this we proposed an initial design to accelerate auto-tuning frameworks and we outlined some research challenges that need to be addressed to realize this.

Discussion Topics

Studying the role of hardware. In this paper, we used a single hardware setup. Given that cloud platforms offer a variety of hardware choices, we believe it is interesting to study if the important knobs vary across different hardware configurations. The models maintained by existing learning-based tuners need to be retrained if the target hardware changes; specifically, new training samples for the new hardware configuration must be obtained [23]. We believe by studying how sensitive the important knobs are to the hardware, we could potentially avoid retraining for new platforms.

Optimizing for composite metrics. In our analysis, we aimed to optimize one metric (such as throughput or average latency) at a time. While this is a first good step, we expect that some deployments might want to optimize for composite metrics. For example, practitioners may want to improve the overall throughput while keeping the operation latencies within a bound. We believe it will be interesting to study if and how the set of important knobs changes when optimizing for composite metrics.

Reliability-aware tuning. Most existing auto tuners aim to find the configuration that provides the highest performance possible. In this pursuit, these tools often compromise on reliability; for example, they may turn off `fsync`, leading to better performance but significantly higher possibility of data loss. We consider this a serious limitation in existing tuners and propose that methods to find the important knobs must also take the target reliability metrics into account.

Acknowledgements

We would like to thank the anonymous reviewers and our shepherd Vasiliki Kalavri for their insightful comments that improved this paper. This work is supported by the National Science Foundation grant CNS-1838733, a Facebook faculty research award and by the Office of the Vice Chancellor for Research and Graduate Education at UW-Madison with funding from the Wisconsin Alumni Research Foundation.

References

[1] Dana Van Aken, Andrew Pavlo, Geoffrey J. Gordon, and Bohan Zhang. Automatic Database Management System Tuning Through Large-scale Machine Learning. In *Proceedings of the 2017 ACM SIGMOD International Conference on Management of Data (SIGMOD '17)*, Chicago, IL, May 2017.

[2] Apache. Cassandra. http://cassandra.apache.org/.

[3] Apache Cassandra. http://cassandra.apache.org/doc/latest/configuration/cassandra_config_file.html#concurrent-reads.

[4] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[5] Leo Breiman, Jerome Friedman, Charles J Stone, and Richard A Olshen. *Classification and regression trees*. CRC press, 1984.

[6] Zhen Cao, Geoff Kuenning, and Erez Zadok. Carver: Finding Important Parameters for Storage System Tuning. In *Proceedings of the 18th USENIX Conference on File and Storage Technologies (FAST '20)*, Santa Clara, CA, February 2020.

[7] Zhen Cao, Vasily Tarasov, Sachin Tiwari, and Erez Zadok. Towards better understanding of black-box auto-tuning: A comparative analysis for storage systems. In *2018 {USENIX} Annual Technical Conference ({USENIX}{ATC} 18)*, pages 893–907, 2018.

[8] Surajit Chaudhuri and Vivek Narasayya. Index Selection Tool for Microsoft SQL Server. In *Proceedings of the 23rd International Conference on Very Large Databases (VLDB 23)*, Athens, Greece, August 1997.

[9] Surajit Chaudhuri and Vivek Narasayya. Self-tuning Database Systems: A Decade of Progress. In *Proceedings of the 33rd International Conference on Very Large Databases (VLDB 33)*, Vienna, Austria, September 2007.

[10] Liu Chu, Eduardo Souza de Cursi, Abdelkhalak El Hami, and Mohamed Eid. Reliability based optimization with metaheuristic algorithms and latin hypercube sampling based surrogate models. 2015.

[11] Brian F Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. Benchmarking Cloud Serving Systems with YCSB. In *Proceedings of the ACM Symposium on Cloud Computing (SOCC '10)*, Indianapolis, IA, June 2010.

[12] Karl Dias, Mark Ramacher, Uri Shaft, Venkateshwaran Venkataramani, and Graham Wood. Automatic Performance Diagnosis and Tuning in Oracle. In *Proceedings of the 2nd Conference on Innovative Data Systems Research (CIDR 2005)*, Asilomar, CA, January 2005.

[13] Songyun Duan, Vamsidhar Thummala, and Shivnath Babu. Tuning database configuration parameters with ituned. *Proceedings of the VLDB Endowment*, 2(1):1246–1257, 2009.

[14] Dmitry Duplyakin, Robert Ricci, Aleksander Maricq, Gary Wong, Jonathon Duerig, Eric Eide, Leigh Stoller, Mike Hibler, David Johnson, Kirk Webb, Aditya Akella, Kuangching Wang, Glenn Ricart, Larry Landweber, Chip Elliott, Michael Zink, Emmanuel Cecchet, Snigdhaswin Kar, and Prabodh Mishra. The design and operation of CloudLab. In *Proceedings of the USENIX Annual Technical Conference (ATC)*, pages 1–14, July 2019.

[15] David A Freedman. *Statistical models: theory and practice*. cambridge university press, 2009.

[16] Stanton A Glantz, Bryan K Slinker, and Torsten B Neilands. *Primer of applied regression and analysis of variance*, volume 309. McGraw-Hill New York, 1990.

[17] Michael Levandowsky and David Winter. Distance between sets. *Nature*, 234(5323):34–35, 1971.

[18] Michael D McKay, Richard J Beckman, and William J Conover. Comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics*, 21(2):239–245, 1979.

[19] Thais Mayumi Oshiro, Pedro Santoro Perez, and José Augusto Baranauskas. How many trees in a random forest? In *International workshop on machine learning and data mining in pattern recognition*, pages 154–168. Springer, 2012.

[20] Adam J Storm, Christian Garcia-Arellano, Sam S Lightstone, Yixin Diao, and Maheswaran Surendra. Adaptive Self-tuning Memory in DB2. In *Proceedings of the 32nd International Conference on Very Large Databases (VLDB 32)*, Seoul, Korea, September 2006.

[21] Clifton D Sutton. Classification and regression trees, bagging, and boosting. *Handbook of statistics*, 24:303–329, 2005.

[22] Tianyin Xu, Long Jin, Xuepeng Fan, Yuanyuan Zhou, Shankar Pasupathy, and Rukma Talwadker. Hey, you have given me too many knobs!: Understanding and dealing with over-designed configuration in system software. In *Proceedings of the Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE '15)*, Bergamo, Italy, August 2015.

[23] 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 *Proceedings of the 2019 ACM SIGMOD International Conference on Management of Data (SIGMOD '19)*, Amsterdam, Netherlands, July 2019.

[24] 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 *Proceedings of the ACM Symposium on Cloud Computing (SOCC '17)*, Santa Clara, CA, September 2017.