

The Case for Automatic Database Administration using Deep Reinforcement Learning.

Authors:

Ankur Sharma, Felix Martin Schuhknecht, Jens Dittrich

Presented By:

Hrushikesh Pappuri

What is the paper about?

The paper discusses the potential benefits of using deep reinforcement learning techniques for automatic database administration. The authors argue that current database management systems (DBMS) require significant human intervention, which can lead to errors, inefficiencies and high costs.

The proposed framework that utilizes reinforcement learning algorithms to automate certain tasks with DBMS, such as query optimization, workload management, and index selection.

The approach is evaluated on a real-world database workload using PostgreSQL and compared with traditional methods such as expert systems and rule-based systems. Results show that the deep reinforcement learning-based approach outperforms traditional methods in terms of both performance and resource usage.

Context of this work

The authors propose an approach that leverages the power of deep reinforcement learning to automate many of the tasks performed by database administrator, including tuning and optimizing database parameters, managing resource allocation, and repairing system failures.

The authors argue that their approach has the potential to improve the efficiency and effectiveness of database administration, while reducing the workload and expertise required by human administrators.

What ML method did they use?

- Deep Reinforcement Learning (DRL), as a primary Machine Learning method.
- The training process in Reinforcement learning does not require any expected outputs.
- The training process is completely driven by rewards that tell the learning whether a taken action leads to a positive or negative result.
- The authors defined a problem environment consisting of four components: Input, Actions, Reward Function, and Hyper Parameters to perform the learning.
- With right instance of these categories, a neural network can be trained for a given optimization goal and a given workload.

How does it utilize and extend to other work ?

Previous research has focused more on rule-based approaches, where a set of predefined rules and heuristics are used to optimize database performance, but they fail to handle the complexity and dynamics of modern databases.

Optimizing database configuration parameters to improve its performance. The agent learns to adjust the database's memory allocation, disk space usage, and other parameters to achieve the best possible performance while minimizing downtime and other disruptions.

Using an agent that can learn and adapt to changing database conditions, the authors hope to achieve improved performance and reliability compared to other approaches.

Example of Atari Game Breakout

The workflow of deep reinforcement learning can be applied to a variety of problem, illustrated by the example of playing the Atari game Breakout and index selection in database administration.

The input is the current state of the game board or workload and index configuration

The set of actions includes the possible movements of the paddle or creating an index on a particular column.

The reward function generates positive rewards if the paddle hits the ball or if an index configuration improves over the previous state.

The learning process starts with random and meaningless predictions but over time, the system learns to make better predictions by positioning the paddle closer to the falling ball or selecting better indexes for query execution.



Figure 1: Breakout on the Atari 2600

Category	Atari Breakout [9]	NoDBA [this work]
Input	Current state of the game board.	Workload and current index configuration.
Set of Actions	Move paddle left, move paddle right, stay	Create an index on a particular column.
Reward	Positive, if the paddle hits the ball.	Positive, if an index configuration improves over previous state.

Table 1: Comparison of Input, Set of Actions, and Reward for Atari Breakout and NoDBA.

Design Advisory Tools

- Classical helper tools for DBAs
- Used by passing a workload file containing a set of SQL queries to come with index recommendations.
- Virtual indexes are introduced, which are just descriptions of hypothetical indexes in form of metadata that is written into the catalog.
- The query is tricked into believing that a variety of indexes exist and considers them in finding the best query plan. (“tricked” means that the query optimizer is given the impression that there are multiple indexes available to use for the query optimization process.)
- When the best query is found, a recommendation for index creation is returned for every column on which an index access occurs in that plan.

NoDBA - Neural Network Input

- The input to the neural network is typically a combination of the encoding of the workload and the current configuration.
- The input is separated into two parts: "workload" and "indexes", which are both fed into the neural network.
- In "workload", the characteristics of the workload are encoded in the form of a matrix of size $n \times m$, where n is the number of queries in the workload and m is the number of columns in the database schema.
- An entry i, j ($i < n, j < m$) of the matrix describes for a query Q_i and a column C_j , the selectivity $Sel(Q_i, C_j)$

$$Sel(Q_i, C_j) = \begin{cases} \frac{\# \text{ selected records of } Q_i \text{ on } C_j}{\# \text{ total records of } C_j} & \text{if } Q_i \text{ selects on } C_j \\ 1 & \text{if } Q_i \text{ does not select on } C_j \end{cases} \quad I_{workload} = \begin{bmatrix} Sel(Q_0, C_0) & \dots & Sel(Q_0, C_{m-1}) \\ \vdots & \ddots & \vdots \\ Sel(Q_{n-1}, C_0) & \dots & Sel(Q_{n-1}, C_{m-1}) \end{bmatrix}$$

$$I_{indexes} = [hasIndex(C_0) \quad \dots \quad hasIndex(C_{m-1})]$$

NoDBA - Set of Actions

$$A = \{\text{create_index_on}(C_j)\}$$

- The index selection problem involves finding the best set of indexes to create on a database table to improve query performance.
- Reinforcement learning is an approach that trains an agent to take actions that lead to better performance.
- The set of possible actions is limited to creating an index on a specific column or not creating any indexes at all.
- Episodic reinforcement learning is used, which means that multiple steps form an episode.
- During each episode, the agent takes one or more actions from the set A (create an index on a specific column or not create any indexes at all).
- The agent receives feedback in the form of rewards based on how well its actions improve query performance.
- The reward function evaluates the impact of a taken action based on how much it improves or degrades performance.
- The agent's objective is to maximize the cumulative reward over the course of an episode, which will lead to the selection of the best set of indexes for the database table.

Reward Function

The reward function rates the impact of a taken action based on how much it improves or degrades performance.

Positive rewards are returned to the agent if query performance improves after creating an index on a column.

Negative rewards are returned if query performance degrades after taking an action.

The specific formula for calculating rewards may vary depending on the problem and goals of reinforcement learning.

$$r(L) = \max\left(\frac{\text{cost}(\emptyset)}{\text{cost}(L)} - 1, 0\right)$$

Evaluation - Setup

- Experiments conducted on desktop PC with specific hardware specs
- Keras-rl used for training, a general reinforcement learning library built on top of Keras
- Keras is a high level neural network API on top of deep learning backends like TensorFlow, Theano, or Microsoft Cognitive Toolkit (CNTK)
- Microsoft Cognitive Toolkit with GPU support chosen as backend for this work.
- Gym Library from OpenAI used for problem environment.
- Optimization of experience replay used in training process to avoid getting stuck in local minima.

Evaluation Setup

- The TPC-H benchmark is a standard benchmark used to evaluate the performance of database systems for decision support applications.
 - It consists of a set of SQL queries that simulate the operations of a typical decision support system, such as generating reports or analyzing trends.
 - It includes various table sizes, called "scale factors", which are used to test the performance of the system under different workloads.
- In this paper, the authors used the schema and data of the TPC-H benchmark in scale factor 1.
 - Scale factor 1 represents a small dataset, with a total size of 1 GB, which is suitable for testing the performance of a single desktop machine.
 - They ran the queries of the benchmark on a PostgreSQL database.

Experiments

Workload	NoIndex [ms]	IndexedAll [ms]	NoDBA [ms]
$W_1 : Q1$	562.143	13.912	38.350
$W_1 : Q2$	530.348	31.868	38.921
$W_1 : Q3$	490.926	82.439	74.666
$W_1 : Q4$	552.060	46.091	37.583
$W_1 : Q5$	545.870	39.811	24.067
$W_1 : \text{Total}$	2681.347	214.121	213.587
$W_2 : Q1$	498.074	127.356	276.205
$W_2 : Q2$	491.517	999.767	193.690
$W_2 : Q3$	548.675	29.630	24.653
$W_2 : Q4$	548.188	25.126	24.617
$W_2 : Q5$	543.516	25.123	24.086
$W_2 : \text{Total}$	2629.97	1207.002	543.251
$W_3 : Q1$	705.850	0.057	1028.729
$W_3 : Q2$	737.364	1.375	977.043
$W_3 : Q3$	728.996	803.244	743.211
$W_3 : Q4$	644.503	0.923	621.37
$W_3 : Q5$	851.598	1065.513	6.602
$W_3 : \text{Total}$	3668.311	1871.112	3376.955

Table 2: Comparison of workload execution times. We compare *NoIndex* (executing the workload without any indexes) with our *NoDBA* (executing the workload with the predicted indexes). Additionally, we show *IndexedAll*, where an index on every column is available.

Experiment

- The paper evaluates the performance of running SQL queries on the LINEITEM table of TPC-H benchmark dataset with and without indexes.

- The workloads consist of SELECT COUNT(*) operations with different WHERE clauses and a fixed set of attributes of LINEITEM.

- The selections are based on randomly chosen columns and perform either equality or range selection with values randomly selected from a list of actually occurring values.

- Workloads W1 and W2 consist of queries that perform up to six selections with a fixed set of attributes of LINEITEM.

- Workload W3 chooses its selections on all attributes of LINEITEM.

- NoDBA system performed as good or even better than having indexes on all columns for W1 and W2 workloads.

- NoDBA's recommended indexes improved the runtime for W3 workload even with selecting only 3 indexes.

- The training time of NoDBA's network was 42 minutes, and individual prediction took only around 20ms.

Query	Selections
Q1	<code>l.partkey < 100, l.supkey < 100, l.linenum = 1, l.discount = 0.02</code>
Q2	<code>l.orderkey < 100000, l.partkey < 10000, l.quantity = 1, l.linenum = 1</code>
Q3	<code>l.quantity = 1, l.partkey < 100000, l.supkey < 1000, l.orderkey < 100000</code>
Q4	<code>l.orderkey < 100000, l.discount = 0.0, l.supkey < 1000, l.linenum = 1</code>
Q5	<code>l.orderkey < 100000, l.partkey < 100000, l.supkey < 10000, l.linenum = 1, l.discount = 0.01</code>

Table 3: Workload W₁

Query	Selections
Q1	<code>l.orderkey < 1000000, l.partkey < 10000, l.supkey < 10000, l.linenum = 1, l.quantity = 1</code>
Q2	<code>l.quantity = 1, l.partkey < 100000, l.orderkey < 1000000, l.linenum = 1</code>
Q3	<code>l.orderkey < 100000, l.partkey < 100000, l.supkey < 100000, l.quantity = 1, l.discount = 0.0</code>
Q4	<code>l.orderkey < 100000, l.partkey < 100000, l.supkey < 100000, l.linenum = 1, l.quantity = 1, l.discount = 0.02</code>
Q5	<code>l.orderkey < 100000, l.partkey < 100000, l.supkey < 100000, l.linenum = 1, l.discount = 0.02</code>

Table 4: Workload W₂

Query	Selections
Q1	<code>l.orderkey < 10, l.supkey < 50000, l.extendedprice < 50000, l_receiptdate < '1993-12-31', l_returnflag = 'A', l_linestatus = 'O'</code>
Q2	<code>l.orderkey < 1500, l.extendedprice < 10000, l_shipinstruct = 'TAKE BACK RETURN', l_receiptdate < '1993-06-30'</code>
Q3	<code>l.supkey < 100, l_shipdate < '1993-01-01', l_receiptdate < '1992-06-29', l_linenum = 4, l_shipinstruct = 'TAKE BACK RETURN', l_shipmode = 'SHIP'</code>
Q4	<code>l.orderkey < 1500, l.supkey < 1000, l_shipdate < '1995-03-31', l_linenum = 4, l_tax = 0.0, l_returnflag = 'N'</code>
Q5	<code>l.supkey < 50000, l.extendedprice < 1000, l_commitdate < '1995-01-28', l_receiptdate < '1992-06-29', l_quantity = 1, l_linestatus = 'O'</code>

Table 5: Workload W₃

Future Work & Conclusion

- Study focus on using deep reinforcement learning for index selection
- Trained a neural network for this task and showed promising results
- Exploring how to extend this work to other areas of DBMS optimization, such as query optimization
- Investigating the trade-offs of using deep reinforcement learning without cost estimates.
- Authors work represents an important step towards improving DBMS performance and automating tedious administration tasks.

Positive Aspects of the Paper

Has the potential to significantly reduce human intervention, lower the error rate, and improve the overall performance of DBMS.

The paper provides a thorough literature review that discusses the current state-of-the-art techniques in database administration and highlights the limitations of existing approaches.

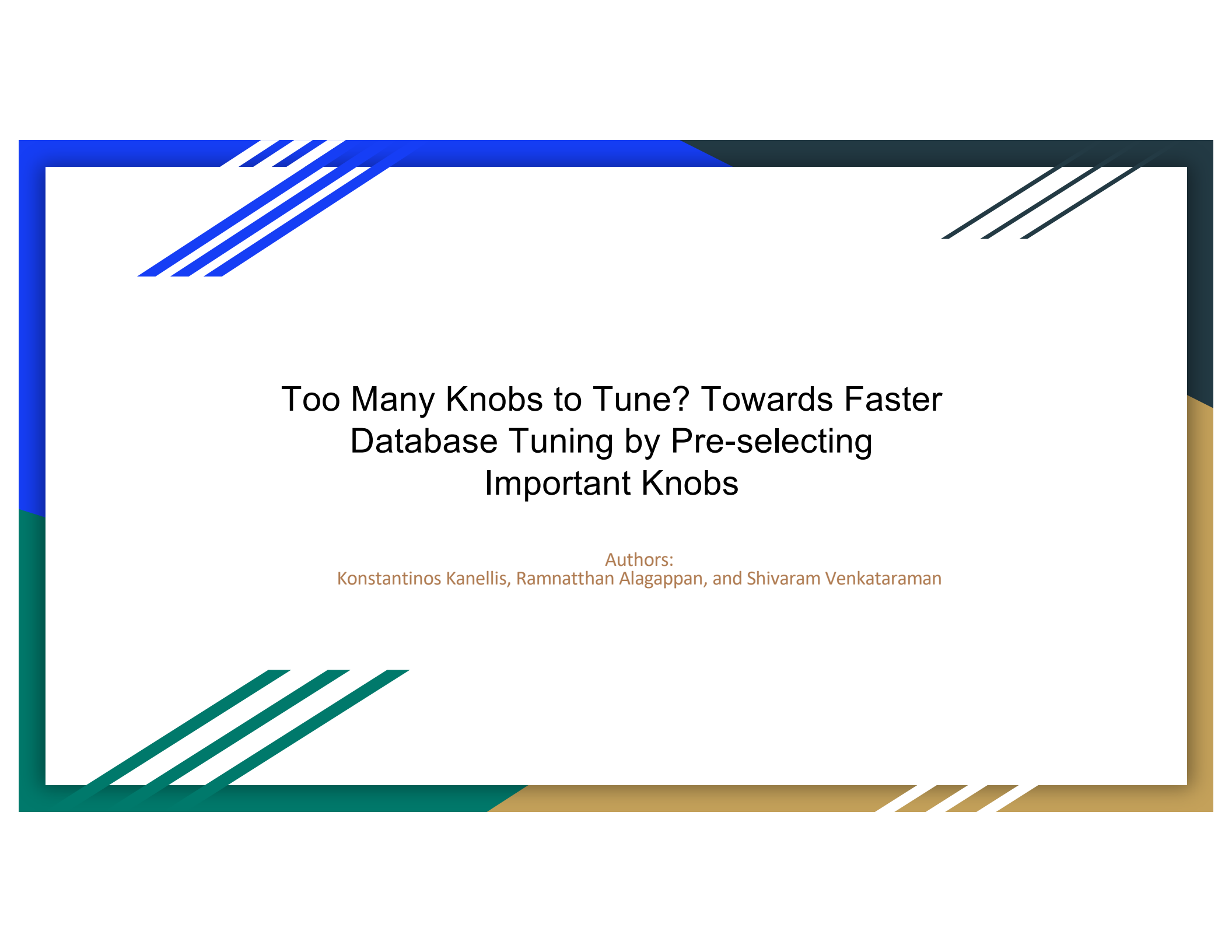
Paper evaluates proposed approach with real-world workload and shows it outperforms traditional methods in terms of performance and resource usage, demonstrating potential for improving database management efficiency.

Negative Aspects of the Paper

- Need for extensive training data
- Not yet to be implemented in a real-world setting.
- The proposed approach may not be applicable to all types of databases or workloads, as it is evaluated only on a specific database using PostgreSQL.

Improving the Proposed Approach with Uncertainty Incorporation

- The proposed approach utilizes reinforcement learning algorithms to automate certain tasks with database management systems.
- However, unpredictability in the database workload can impact the performance of the system.
- Incorporating uncertainty into the reinforcement learning algorithm can improve the system's ability to handle unexpected spikes in workload.



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

Authors:
Konstantinos Kanellis, Ramnatthan Alagappan, and Shivaram Venkataraman

What's the purpose & What ML methods did they use

Use statistical techniques to pre-select a small set of important knobs that have significant impact on performance

ML method they used is Lasso regression, a type of machine learning to identify the most important knobs based on their correlation with performance metrics

The results show the proposed approach significantly reduces tuning time and improves performance

Experimental Setup

- The authors studied two popular database systems, Cassandra and PostgreSQL, to better understand the impact of different configuration knobs on their performance.
- They used the YCSB-A and YCSB-B workloads from the Yahoo! Cloud Serving Benchmark (YCSB) suite to simulate different usage scenarios.
- The authors identified a set of performance-related knobs for each system, tweaking these knobs to generate 25,000 unique samples per system-workload pair.
- They standardized the values of each knob to ensure that the range of values did not influence the importance of any individual knob in their regression models.
- Using random forest regression models, the authors were able to identify the most important knobs for each system and workload.

Important Knobs Identified by Lasso

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.

- Table 1 summarizes the performance of Apache Cassandra and PostgreSQL databases under different knob configurations for the YCSB-A workload.
- The table lists the number of samples and knobs used for each configuration, as well as the throughput, read latency, and write latency for each database system.
- The "Baseline" configuration represents the default settings for each database system, while the "Validation" configuration includes only the most important knobs identified by the study.
- Apache Cassandra achieved a higher throughput than PostgreSQL for all knob configurations tested in the study.
- By adjusting the top 5 most important knobs, Apache Cassandra achieved a throughput of 99.36% of the baseline, while PostgreSQL achieved a throughput of 99.10% of the baseline.

Experimental Setup - Collect Data

- 30 machines with identical hardware specifications were used to parallelize sample collection
- Each machine ran one experiment at a time and was part of the CloudLab infrastructure
- The database system and YCSB clients were run on the same physical machine but isolated on separate CPUs
- Each machine had a 10-core Intel Xeon Silver 4114 CPU with 64 GB of memory and used a 480-GB SSD for storage
- Each experiment took approximately 9 minutes to run
- Collecting 25K samples for a single system-workload pair required approximately 3750 node-hours (or a bit over 5 days when using 30 nodes)
- The information provides insight into the scale of the study and the resources required to collect data for analysis

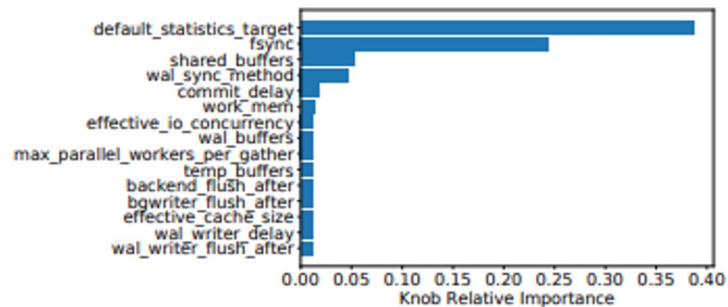


Figure 4: YCSB-A

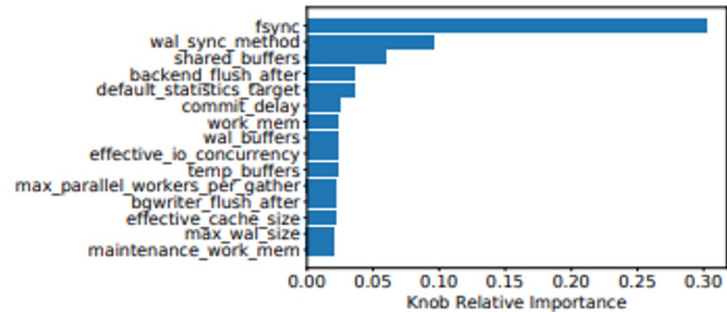


Figure 5: YCSB-B

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

- Figure 4 shows the relative importance of different knobs for Apache Cassandra and PostgreSQL database systems when evaluated with YCSB-A workload
 - o X-axis represents knobs and y-axis represents their relative importance based on a random forest model
 - o A handful of knobs are most important for performance optimization under this workload

- Figure 5 shows the impact of different knob values on system throughput for Apache Cassandra and PostgreSQL database systems when evaluated with YCSB-A workload
 - o X-axis represents knob values and y-axis represents throughput in operations per second (ops/sec)
 - o Changing certain knobs can have a significant impact on system throughput, while changing other knobs has little to no impact

Positive Aspects

This paper is well-written and clearly presents the problem, proposed solution, and evaluation results, making it easy for readers to understand and follow.

The authors evaluate their approach on several real-world workloads, demonstrating its effectiveness in reducing tuning time and achieving good performance.

Negative Aspects

Lack of open source implementation

Authors compare the proposed approach to traditional methods for database tuning but the authors do not compare their approach to that recently proposed machine learning-based approach that address the same problem.

Conclusion

This study has shown that tuning just a few knobs can lead to optimal performance in database systems. This finding is consistent across different workloads and database systems.

The proposed design to accelerate auto-tuning frameworks can potentially improve efficiency in knob tuning, but there are still research challenges to be addressed.

One area of research that needs to be explored is the role of hardware in knob tuning. The importance of specific knobs may vary across different hardware configurations, and this needs to be studied to avoid retraining existing models.

Additionally, while this study focused on optimizing one metric at a time, future research should explore composite metrics, where practitioners may want to improve overall throughput while keeping operation latencies within a bound.

In summary, this study provides valuable insights into knob tuning for database systems and highlights important areas for future research.