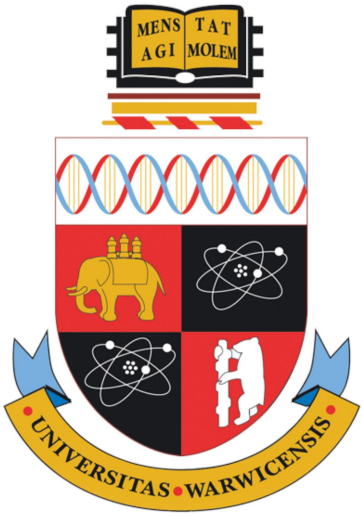
OLTP workload mapping for database tuning

CS310 Computer Science Project

Valentin Kodderitzsch



Supervised by Prof Peter Triantafillou

Department of Computer Science

University of Warwick

2023

Abstract

This dissertation aims to reduce the sampling bottleneck in the context of OLTP database tuning. Since the inception of the database tuning research landscape in the 1970s, there have been numerous approaches to achieve faster overall database performance. The latest research efforts rely mostly on machine learning-based tuning agents which require exhaustive sampling and offline training before they can be deployed. As such, there exists an incentive to reduce the sampling time to allow for a faster end-to-end tuning process.

To this end, a novel similarity score, sampling function, and ensemble model have been developed for this dissertation. The rationale behind each design choice will be elucidated before proceeding to experimentally determine the effectiveness of the proposed solutions. In the end, the ideas presented in this dissertation were able to significantly reduce the sampling bottleneck whilst still returning competitive database settings.

Keyword list: Database tuning, workload mapping, machine learning, sampling techniques, ensemble model, TPC-C

Contents

Acknowledgement

Firstly, I would like to express my gratitude towards Professor Peter Triantafillou for accepting me for this dissertation and for his guidance throughout the project. Without your continuous insights and engaging feedback, this project would not have been possible.

Secondly, I want to sincerely thank George Barbulescu for the initial onboarding and for sharing his code base with me. This was an invaluable contribution to this project. Without George's code base the ideas I developed and experimentally evaluated would have fallen short due to time constraints.

Thirdly, I would also like to express my gratitude to Meghdad Kurmanji for his seemingly never-ending patience. I am very grateful for your presence in all our meetings with Peter and your insights.

Finally, I would to thank the open-source community and online forums for their publicly available knowledge. I would also like to thank my friends and family for their continuous mental support and encouraging words.

1 Introduction

The following section presents the motivation behind this dissertation. Key terminology will be briefly explained before presenting the research statement. Finally, the structure of the remaining report will be outlined.

1.1 Motivation

The motivation of this dissertation is to add to the current research landscape for database tuning in the context of workload mapping. Database tuning as described Pavlo et al. in 2019 [35] is the process of systematically choosing the best database settings in order to "improve a DBMS's (database management system) operations based on some objective function (e.g. faster execution, lower costs, better availability)".

There exists a plethora of settings and low-level database architecture configurations which can be manipulated. Additionally, there have been numerous tuning approaches since the inception of the field in the 1970s [35]. Both of these will be discussed in more detail in the background section. However, generally speaking one of the main motivations for database tuning is that it is a $\mathcal{NP}$-hard problem [46]. As such, leveraging machine learning (ML) models to approximate the problem becomes a promising proposition for the computer science community to explore.

Consequently, the focus of this dissertation will be database performance tuning via machine learning models. As one might expect the tuning problem becomes a *regression problem* where the independent variables are the database settings and the dependent variable is the database performance. Once the regression line has been fitted the most competitive database settings which ought to be recommended are the $argmax$ values of the regression line. There already exists a multitude of machine learning based database tuning managers such as OtterTune [46] as developed by the Carnegie Mellon University in 2017 or BTSTR [4] which has been developed by the University of Warwick in 2022.

One of the major drawbacks of tuning managers such as OtterTune or BTSTR is their data sampling collection process which takes about 10 hours. Samples are collected to tune the database for a specific use case and need to be re-sampled if the use case changes. However, if the use cases are similar it is debatable if an exhaustive sampling approach

is required. Based on this intuition a formal research statement will be introduced in the objectives section.

1.2 Terminology

Before moving on to the objectives section some key terminology needs to be briefly explained. Said terminology will aid with the understanding of the research statement, the state of the current research landscape, and the novel ideas presented in this project. The following terms will be defined and used throughout the rest of the dissertation:

1. Database Performance
2. Database Settings
3. OLTP
4. Workload
5. Workload Characterization
6. Sampling
7. Workload Mapping

1.2.1 Database Performance

Database performance can be measured using numerous metrics. The most common are throughput and latency. Latency measures the time it takes to execute a data read or write command against the database. Throughput is proportional to latency and measures the number of successfully executed commands within a certain time window. Usually, throughput is measured as requests per second or per minute. The lower the latency the higher the throughput. Database performance is improved if the latency is reduced or the throughput is increased compared to the baseline measurement.

1.2.2 Database Settings

There exist numerous database settings, however, for the purpose of this dissertation DBA (database administrator) level settings will be used as further explained in the design section. Database settings are tuneable if they can be manipulated in the configuration file of the database. As such, physical design changes such as manipulating

indexes or views are not considered tuneable database settings for this project. The same applies to query processing techniques such as selectivity estimations or hardware changes to the database, e.g. physically increasing the amount of RAM of the server. Therefore, tuneable database settings are configurations which determine the amount of RAM, CPU and logs allocated to the database given the hardware profile of the server on which the database runs.

1.2.3 OLTP

OLTP stands for online transaction processing and is a type of relational database. OLTP databases are "optimized to process queries that may touch a small part of the database and transactions that deal with insertions or updates of a few tuples per relation" (page 1103) as explained by Elmasri et al. [9]. In other words, OLTP databases perform both read and write commands to the database and can thus change its size. Typical applications of OLTP databases include managing airline reservations or the stock of e-commerce platforms.

OLTP databases are the counterpart to OLAP (online analytical processing) databases. As opposed to OLTP databases, OLAP databases are concerned with analysing complex data from data warehouses. Mainly read-only commands are executed against an OLAP database meaning that the size of the OLAP database seldomly changes.

1.2.4 Workload

The workload is the specific use case for which samples are collected. A workload consists of multiple transactions which may affect the state of the database. A transaction as defined by Elmasri et al. [9] is a "logical unit of database processing [...] typically implemented by a computer program that includes database commands such as retrievals, insertions, deletions, and updates" (page 745). As such, a read-heavy workload for instance might include numerous read-only transactions coupled with some write transactions which will only slightly update the state of the database. On the other hand, a write-heavy workload will consist of many write-only transactions combined with some read transactions.

The notion of a workload is relevant for database tuning as a machine learning tuning agent will optimise the database settings for a specific workload. The intuition here

is that read-heavy workloads (e.g. analysing historic weather data) require different database settings compared to write-heavy workloads (e.g. updating the inventory of an e-commerce platform) to obtain optimal performance values.

1.2.5 Workload Characterization

Workload characterization as explained by Van Aken et al. [45] is the process which "identifies a subset of DBMS metrics that best capture the variability in performance and the distinguishing characteristics of a given workload" (page 1246). In other words, workload characterization is about finding a set of database statistics (e.g. latency) that best capture the database's performance while running a specific workload to make it identifiable. As such workload characterization is exclusively concerned with database performance at a workload level, whereas general database performance can range from a micro (e.g. single commands) to a macro (e.g. sets of workloads) perspective.

1.2.6 Sampling

Sampling is the process of collecting the data points for a specific workload. As hinted at in the database settings section the performance values will be determined by the executed workload if the hardware profile of the database server remains the same. Consequently, the sampling process can also be called the workload execution process or the benchmarking process to obtain workload specific performance data. The sampling process is part of the workload characterization process. However, the emphasis is less on uniquely identifying a workload and more on the actual process of running benchmarks and collecting data points.

1.2.7 Workload Mapping

Workload mapping is the process of matching the current workload to previously seen workloads. As such, workload mapping is concerned with determining the similarity between multiple workloads via their characteristics. Workload mapping is a preprocessing step which can help optimise subsequent operations in the machine learning based database tuning pipeline.

1.3 Research Statement

As briefly mentioned in the motivation section one of the major drawbacks of the current machine learning based tuning managers is their initial data sampling collection process. Said sampling process is very time expensive and is limited to one workload only. As such this dissertation aims to investigate the following ***research statement:***

> How can the sampling bottleneck be reduced when mapping between workloads in the context of OLTP database tuning?

This means that there will be a workload A and a workload B. Workload A is the initial workload and the goal is to reduce the sampling process for workload B. Additionally, the machine learning model for workload B should also be able to make competitive database setting recommendations for workload B.

1.3.1 SMART objectives

The aforementioned goals can be phrased as SMART (specific, measurable, achievable, time-bound) objectives [1] as suggested in the CS352 module. As such, meeting any of the following SMART objectives constitutes a direct contribution to the current research landscape as little research has been done yet in the context of reducing the sampling bottleneck. For the purpose of this dissertation, the following SMART objectives are defined:

1. To decrease the convergence time of the new sampling method by at least 10% compared to the original sampling method, by May 2nd 2023
2. To decrease the latency for the new workload by at least 10% compared to the original solution, by May 2nd 2023

It is important to note that 10% is an arbitrary number and the emphasis is on a notable change rather than the actual number of 10%.

2 Background

The following sections will first explain the motivation and purpose behind database tuning. Then, the most common database tuning approaches will be elucidated followed by an in-depth explanation of two selected database architectures. An explanation of different sampling techniques and database settings will also be provided.

2.1 Motivation for Database Tuning

As explained by Pavlo et al. [35] the history of database tuning first started in the 1970s with self-adaptive database designs [14] and in the last five decades DBMS tuning techniques have changed dramatically. However, the motivations behind end-to-end automated tuning remain largely the same.

Firstly, there is a need to perform DBMS tuning as most out of the box database settings are generic and have not been optimised for any specific use case. The 2023 default MySQL configuration for instance assume that the local server only has 512 MB of RAM [33] which is far less then most commercial servers. As such, by simply using more of the available RAM performance increases can be expected.

Secondly, the total number of tunable settings have been steadily increasing with each new database version release. Van Aken et al. for instance found in their 2017 paper [46] that the total number of configuration knobs for the PostgreSQL DBMS have tripled in a span of 15 years. Thus, advances have been made in the database tuning research community such as by Kanellis et al. in 2020 [22] to reduce the total number of PostgreSQL configuration knobs from around 170 to five. The premise of Kanellis' paper is that "tuning just five knobs can achieve 99% of the performance achieved by the best configuration that is obtained by tuning many knobs".

Thirdly, the set of all tunable settings is not only large but also highly interdependent. This means that increasing one setting might negatively affect another setting and vice versa. Therefore, it has been argued by authors such as Mukkamala et al. [29] and Raghavan et al. [19] since the 1980s that DBMS tuning is a $\mathcal{NP}$-hard problem. $\mathcal{NP}$-hard means that the problem is "at least as hard as any $\mathcal{NP}$-problem" [47] meaning that the problem cannot be solved efficiently in polynomial time. As previously mentioned,

this creates an incentive to develop approximation-based ML models to solve the problem at hand.

Finally, as shown in the work presented by Van Aken et al. [46] a database can be tuned for a specific use case. However, these specific settings are not always re-usable meaning there is a need for continuous tuning as shown in figure 1. Thus it is crucial for any database administrator (DBA) to carefully reconsider the database settings when the workload changes to ensure optimal performance.

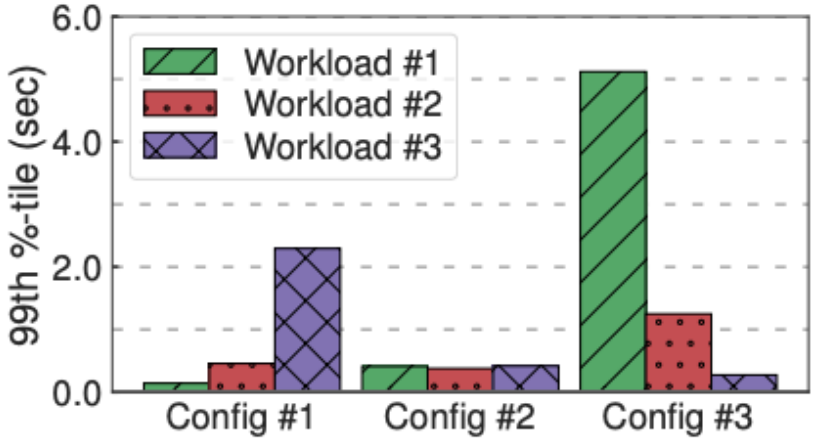


Figure 1: Non-reusable settings - Taken from [46]

To conclude, databases are meant to be tuned as their default configurations are non-competitive. However, the actual tuning process is a non-trivial task and there exists the additional incentive to solve the DBMS tuning problem in a efficient manner since hiring DBAs is costly.

2.2 Types of Database Settings

As mentioned in the introduction section there exists a deluge of possible database settings which can be tuned. As summarised by Pavlo et al. in 2019 [35] there exist four types of database settings that have been tuned historically.

The first type is concerned with the physical database design which includes indexes, partitions and clusters. Said settings were first tuned in the 1970s. The most notable papers are by Hammer et al. about self-adaptive index selection [15] and heuristic approaches to partitioning [16].

The second type are "knob configurations" or simply database settings that can be manipulated via a configuration file. Many these settings require a database restart and they manipulate the databases memory allocation such as caching policies or the length of logs. For the purpose of this dissertation only "knob configurations" will be considered as they can be directly controlled by external tuning managers.

The third type of settings are the actual hardware profile of the database. Depending on the server the built in RAM and CPU performance might be drastically different. As such, physically upgrading the RAM or CPU of a server can result in major performance increases.

The last type of database settings are query plan hints. Here a DBA can specifically arrange the ordering of joins for instance such that certain compiled queries are optimised. A DBA also has the possibility to determine heuristics when query plan is optimised.

2.3 Non Machine Learning Based Tuning

Before the 2010s the two main DBMS tuning approaches were either heuristic or cost-based [35]. Here it is also important to highlight that many historic tuning architectures have been tools to help a DBA make decisions rather than end-to-end tuning architectures that can replace a DBA.

One example of such a tool is IBM's DB2 Performance Wizard tool [26] which was released in the early 2000s. IBM's tool asked the DBA a set of questions about the use case of their database and then returned some best practice configurations. IBM's tool is a classic example of a heuristic-based approach that works better than not tuning the database at all. However, this approach does not take the exact workload and database specifications into consideration. As such, better performance increases can be achieved by providing more specific configurations.

An example of a cost-based database setting recommendation system is Microsoft's AutoAdmin tool [5] from 1998. The advantage of said tool is that it was able to estimate the utility of an index based on the Microsoft SQL Server's internal cost models for its query planner. However, cost models do not apply well to database "configuration

knobs" because of their interdependence. Therefore, other methods must be explored when it comes to database configuration tuning.

2.4 Machine Learning Based Tuning

As previously mentioned, there exists an incentive to use ML-based database tuning systems due to the complexity introduced by the interdependence of database settings. When it comes to ML-based database tuning architectures Pavlo et al. [35] argue that there exist two classes, namely internal and external systems.

Internal systems such as NoisePage [36] are designed with an ML-based tuning agent built into the DBMS architecture. As such, the tuning agent is more exposed to the database's internal statistics and can collect more information. Additionally, a built-in tuning manager can control more low-level settings which may result in better performance values. However, the major drawback of internal systems is that they either require a rebuilt of existing DBMS solutions or the development of a new DBMS altogether.

The alternative to internal systems are external ML-based tuning managers which treat the DBMS' architecture as a black box. Instead of being a built-in component the ML agent collects data about the database's performance via its API or third-party monitoring tools. Additionally, the ML agent needs to have the capability to update database settings and potentially restart the DBMS such that said settings become active. Unavailability due to restarts and lack of low-level control are two of the main disadvantages of external ML-based tuning managers. However, external tuning managers have become very popular in recent years with the most cited ones being OtterTune [46], CDBTune [48], BestConfig [52] and ResTune [49].

2.5 Selected Architectures

For the purpose of this dissertation the OtterTune [46] and BTSTR [4] architectures are the most relevant. Reason being that BTSTR is inspired by OtterTune and the novel ideas developed in this dissertation are built on top of BTSTR architecture. Thus, both will be explained in the following two sections.

2.5.1 OtterTune

With over 520 citations on Google Scholar [12] since its publication in 2017, the OtterTune paper written by Van Aken et al. [46] is undoubtedly one of the most popular papers in the domain of automated database tuning. As such, it is necessary to understand the fundamental ideas proposed in this paper as the majority of research done in this domain builds on top of it.

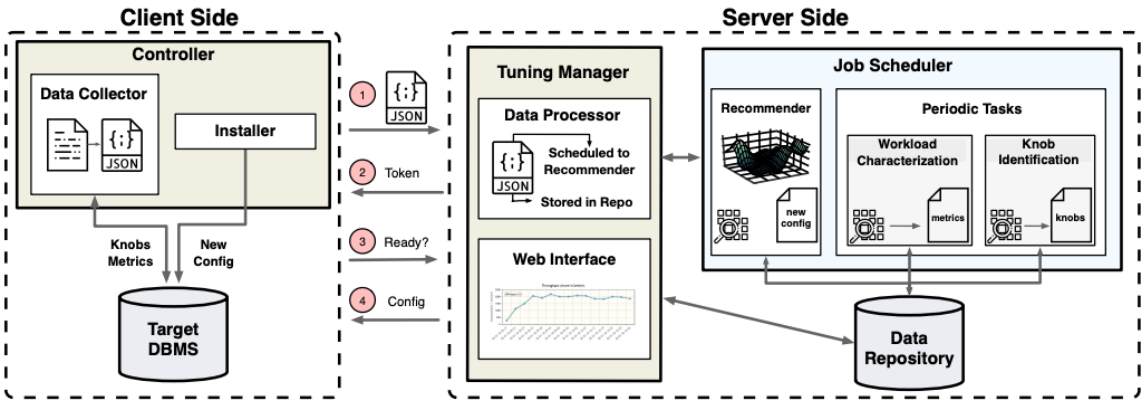


Figure 2: OtterTune Architecture [35]

The OtterTune architecture is an external ML-based system that enables automated end-to-end database tuning. To this end, OtterTune requires an initial offline training phase where it builds a repository of known workloads before it can deal with any unseen workloads. Another characteristic of OtterTune is that it also sorts out redundant database settings in its ML pipeline. This is a noteworthy step because most subsequent solutions do not sort out redundant database settings and instead rely on the results provided by OtterTune.

Besides the DBMS which is meant to be tuned, there exist three main entities in the OtterTune architecture as shown in figure 2. The first one is the controller which interacts with the target DBMS by updating database setting and recording performance values. The second entity is the tuning manager which is the middle-men between the controller and the job scheduler. Finally, there is the job scheduler which is the ML-based tuning agent of the system.

OtterTune's entities follow an iterative workflow where recommendations are made until the user is satisfied with the performance increase. First, the controller passes the

database performance values for a given workload to the job scheduler via the tuning manager. At the same time the tuning manager displays the current status to the user so that they can monitor progress. Once the job scheduler received the database performance values, it characterizes the current workload and identifies the most important database settings that are currently used. Based on the data from its offline training repository combined with the current performance values, the ML-based tuning agent will make a new database setting recommendation. Said recommendation will then installed by the controller and based on the corresponding results OtterTune will continue to make new recommendations. This cycle stops once the user is satisfied.

2.5.2 BTSTR

The BTSTR [4] architecture was developed by Barbulescu et al. in 2022 at the University of Warwick. As seen in figure 3 the overall architecture shares many similarities with the OtterTune architecture. Both architectures rely on offline training data coupled with an iterative approach to recommend new database setting. However, there are also some noticeable differences.

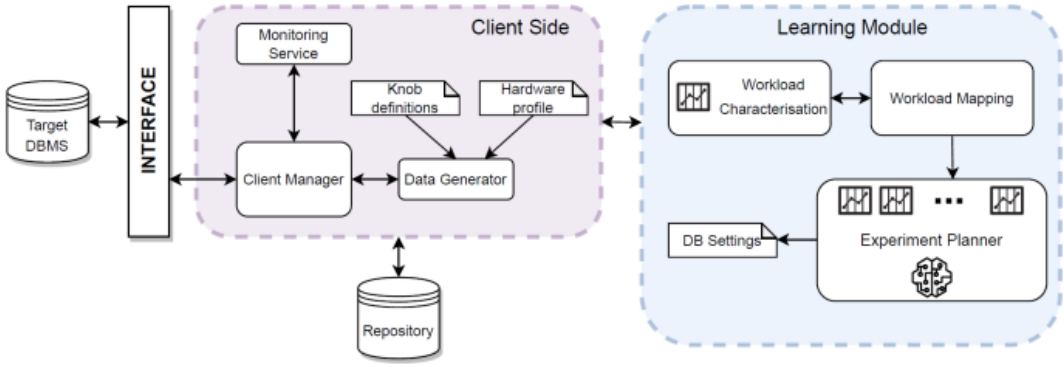


Figure 3: BTSTR Architecture [4]

The main difference is that the BTSTR architecture leverages Bayesian optimisation-inspired techniques. More precisely, it implements a novel ensemble model which allows past tuning sessions to be captured in a lightweight manner. As such, the BTSTR architecture avoids long convergence times and can focus on exploring new recommendations instead. Historic tuning sessions are captured by reusing the model which has already been trained on the historic data. Thus, no data is sent and instead the ensemble model

captures both historic and current information via its constituent models which serve as a proxy for past and current data.

Additionally, the BTSTR architecture has updated the experiment planner. Instead of relying on the Euclidean distance to evaluate potentially competitive and unseen database settings, the BTSTR architecture builds upon its new ensemble model. As such, the database optimisation process is guided by a new expected improvement function which ultimately leads to more competitive database settings than the OtterTune solution.

2.6 Sampling

The majority of ML-based database tuning architectures rely on an initial sampling and training phase. To ensure the highest possible sampling quality each architecture must choose a sampling technique. Said sampling technique is then applied to the domain of all possible database settings resulting in a subset of database settings to be evaluated.

The most popular sampling technique as argued by Kanellis et al. [22] is the Latin Hypercube Sampling (LHS) technique [27]. Other popular sampling techniques are Sobol [20] and Halton [30]. However, LHS seems to be the most effective as argued by Kanellis et al. and it has also been used in the OtterTune architecture.

Latin Hypercube Sampling works by splitting the range of each database setting into n equal size blocks where n denotes the number of desired sample points. Then exactly one point is selected at random from each block which ensures that the entire domain is uniformly covered.

3 Methodology

In the methodology section the research approach and the evaluation baselines will be explained. Additionally, the main hypothesis of this dissertation will be presented coupled with the corresponding hypothesis test. The null hypothesis for this dissertation is that the similarity scores between the original and new workload are the same when comparing the pre-and-post sampling similarity scores. The alternative hypothesis is that the pre-and-post sampling similarity scores cannot be used interchangeably.

3.1 Research Approach

As indicated in the background section, the general database tuning research landscape is quite broad. There exist a multitude of subtopics to investigate and thus narrowing down the focus of the research area is crucial for a successful project. Since this dissertation was carried out between the end of September 2022 and the beginning of May 2023, the overall life cycle of this dissertation was only about seven months.

Due to the short-lived nature of this project the initial research focus was decided at the very start. Said focus was chosen to be transfer learning between changing workloads. However, throughout the experimental phase of the project the sampling process was identified as the main bottleneck as opposed to model training time. Consequently, the research focus was adjusted to be an investigation to reduce the sampling bottleneck as opposed to transfer learning. More concrete results and explanations regarding model training times can be found in the results section.

Additionally, it is important to keep in mind that this project is meant to be a continuation of Barbulescu's work [4] from 2022 at the University of Warwick. However, as already mentioned in the specification report from September 2022 the goal of this project is not to re-create an improved end-to-end automated database tuning manager. Instead the objective is to build an architecture on top of an existing tuning manager with the sole purpose of reducing the sampling bottleneck. Incorporating this new architecture into an automated end-to-end tuning manager is outside of the scope of this research project and will be discussed in the future work section. However, the details of the novel workload mapping feature will be presented in the implementation section. Barbulescu's work will serve as a baseline and the novel workload mapping

feature will be evaluated and discussed in the results section.

3.2 Hypothesis

The main hypothesis for this thesis is that it is possible to characterize a workload before the sampling process. Once such a pre-sample characterization has been achieved it will be possible to compute a similarity score between the original and the new workload. In the end, the proposed solution to reduce the sampling bottleneck will then rely on said similarity score.

	Original Workload	Novel Workload	Similarity Score
Pre-sampling	w_o	w_n	$\hat{\theta}$
Post-sampling	W_o	W_n	θ

Table 1: Workload Characterization Notation (workload, old, new, similarity score)

Before explaining the hypothesis in more detail it is important to specify some notation as summarised in table 1. Capital letter W denotes a post-sampling workload, i.e. a workload where all samples have been collected. As such any workload W can be characterized by its individual run-time statistics as well as the regression line fitted over all run-time statistics. Lower case w denotes a pre-sampling workload which means that no "(database setting, performance value) pairs" have been sampled for said workload w yet. However, in the design and implementation section a novel approach will be introduced to allow for an adequate pre-sampling workload characterization. The subscripts o and n denote original and new. As this dissertation is concerned about workload mapping there need to be at least two workloads. For the sake of simplicity this dissertation is concerned with exactly two workloads only, i.e. the original workload o and the new workload n which is the destination of the mapping. The variable θ denotes the similarity score between the two post-sampling workloads W_o and W_n. The pre-sampling counterpart to θ is $\hat{\theta}$ and it denotes the similarity score between the pre-sampling workloads w_o and w_n. The details of how both similarity scores are computed will be presented in the design and implementation sections.

3.3 Hypothesis Testing

As previously mentioned the main hypothesis is that $\hat{\theta}$ and θ are highly similar and thus the sampling bottleneck can be reduced. One way to formalise this hypothesis is by introducing a null H_0 and alternative H_1 hypothesis. The most straight-forward argumentation would to assume that the null hypothesis is equal to the mean of all $\hat{\theta}$ being the same as the mean of all θ. The alternative would be that said means are not the same. In other words, on average the pre-and-post sampling similarity scores $\hat{\theta}$ and θ are the same. The following equation (1) captures this simple hypothesis which will require a two-tailed test.

$$\begin{aligned} H_0 &: \frac{1}{n}\sum_{i=0}^{n}\hat{\theta}_i = \frac{1}{n}\sum_{i=0}^{n}\theta_i \\ H_1 &: \frac{1}{n}\sum_{i=0}^{n}\hat{\theta}_i \neq \frac{1}{n}\sum_{i=0}^{n}\theta_i \end{aligned} \tag{1}$$

However, as further explained in the results section, the sampling process for a single workload W takes about 10 hours. To empirically test the hypothesis from equation 1 at least $n = 30$ samples need to be collected such that the central limit theorem (CLT) starts to apply [25]. If $n = 30$ samples were to be collected with each sample taking about 10 hours, then the entire sampling process would take about 12 uninterrupted days. Due to the time constraints of this dissertation this was deemed to be an inappropriate approach.

As previously mentioned, however, this dissertation is only concerned with mapping between exactly two workloads. Consequently, the number of samples n can be drastically reduced to $n = 9$ by reusing samples W. Reusing samples becomes a combinatorics problem, i.e. an unordered sampling problem without replacement to be more specific. As such, by setting $n = 9$ and choosing two workloads it is possible to obtain $\mathrm{C}_2^9 = 36$ workload pairs resulting in 36 different θ values. This approach only takes four days as opposed to 12 and the same logic also applies for pre-sampling so $\hat{\theta}$.

3.3.1 Welch's t-test

The hypothesis test deemed most appropriate for the hypothesis presented in equation 1 is Welch's t-test [44]. Firstly, because both the population variance and the population

mean are unknown and thus a t-test needs to be chosen. Secondly, because assuming that both population variances are not equal is a less restrictive assumption. Otherwise, a pooled two-sample t-test might have been chosen. Based on Welch's t-test the critical value approach (or the p-value approach) can be used to accept or reject H_0. The test statistic T from Welch's t-test is defined in equation 2 where X denotes the pre-samples similarity score $\hat{\theta}$ and Y denotes the post-sampling similarity score θ. So $\mu_X - \mu_Y$ (which is equal to 0) is the same as H_0 in equation 1, i.e. denoting that the population means are the same. $\bar{X}$ and $\bar{Y}$ denote the means obtained from the actual data set at hand, i.e. they are the observed sample means. S^2 denotes the the observed sample variance and n, m denote the number of observed samples which should both be $\mathrm{C}_2^9 = 36$.

$$T = \frac{(\bar{X} - \bar{Y}) - (\mu_X - \mu_Y)}{\sqrt{\frac{S_X^2}{n} + \frac{S_Y^2}{m}}} \tag{2}$$

T, as described in equation 2 will approximately follow a t-distribution with r degrees of freedom. The degrees of freedom are determined by equation 3. However, r needs to be an integer so it is common to take the floor of r, so $\lfloor r \rfloor$.

$$r = \frac{\left(\frac{s_X^2}{n} + \frac{s_Y^2}{m}\right)^2}{\frac{(s_X^2/n)^2}{n-1} + \frac{(s_Y^2/m)^2}{m-1}} \tag{3}$$

Once T has been computed it will serve as the test statistic. To compute the rejection region first the degrees of freedom r need to be computed, and then a significance level α needs to be chosen. Then a t-distribution table such as table 2 can be consulted. If a significance level of $\alpha = 0.05$ is chosen and based on the observed $\hat{\theta}$ and θ values $T = 3.5$ and $r = 55$ are computed[1], then equation 4 needs to be consulted. If equation 4 holds then H_0 will be rejected in favor of the alternative hypothesis. Equation 4 is derived from the symmetry of the t-distribution. Since $T = 3.5 > 2.004$ the null hypothesis H_0 needs to be rejected in this example. A pictorial description of equation 4 is shown in figure 4 which is taken from [44]. Ideally, for the purpose of this dissertation it holds that $T < t_{\alpha/2,r}$, i.e the test statistic T is between the two rejection regions meaning that the two means of $\hat{\theta}$ and θ are the same.

[1]These values have been chosen for demonstration purposes only

$$T > t_{0.025,55} = 2.004$$
$$T < t_{0.025,55} = -2.004 \tag{4}$$

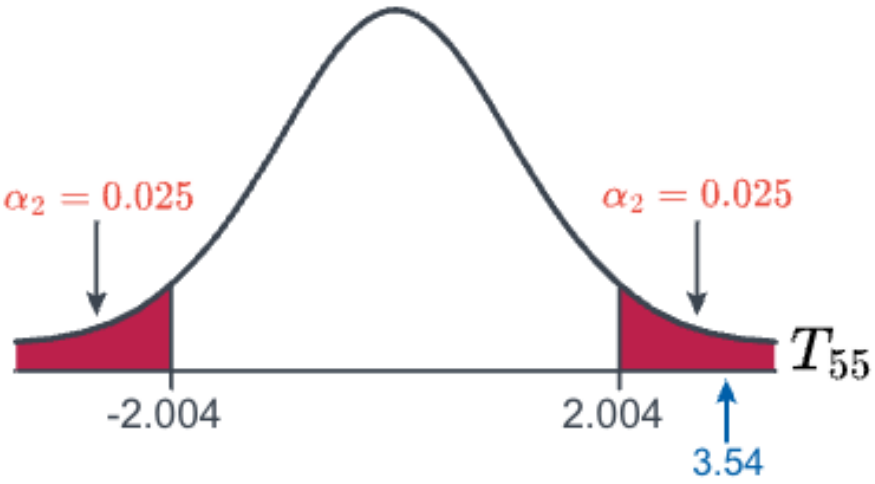


Figure 4: Example rejection region t-test

	Level of Significance for One-Tailed Test								
	.25	**.20**	**.15**	**.10**	**.05**	**.025**	**.01**	**.005**	**.0005**
	Level if Significance for Two-Tailed Test								
df	**.50**	**.40**	**.30**	**.20**	**.10**	**.05**	**.02**	**.01**	**.001**
1	1.000	1.376	1.963	3.078	6.314	12.706	31.821	63.657	63.662
2	.816	1.061	1.386	1.886	2.920	4.303	6.965	9.925	31.599
3	.765	.978	1.250	1.638	2.353	3.182	4.541	5.841	12.924
4	.741	.941	1.190	1.533	2.132	2.776	3.747	4.604	8.610
5	.727	.920	1.156	1.476	2.015	2.571	3.365	4.032	6.869
10	.700	.879	1.093	1.372	1.812	2.228	2.764	3.169	4.587
20	.687	.860	1.064	1.325	1.725	2.086	2.528	2.845	3.850
30	.683	.854	1.055	1.310	1.697	2.042	2.457	2.750	3.646
40	.681	.851	1.050	1.303	1.684	2.021	2.423	2.704	3.551
50	.679	.849	1.047	1.299	1.676	2.009	2.403	2.678	3.496
100	.677	.845	1.042	1.290	1.660	1.984	2.364	2.626	3.390
∞	.674	.842	1.036	1.282	1.645	1.960	2.326	2.576	3.291

Table 2: Percentage Points of the t Distribution [17]

3.4 Evaluation Metrics

Apart from testing whether the hypothesis presented in this dissertation holds, it is also crucial to evaluate whether the novel workload mapping feature actually provides any competitive database settings. Otherwise, it would be possible to meet the goal of reducing the sampling bottleneck at the cost of providing useless database settings to the

user. This is obviously an undesirable approach and needs to be addressed. Therefore, the goal will be to strike a balance between reducing the sampling bottleneck whilst still providing competitive settings.

3.4.1 Baseline

Competitive database settings can be defined as settings that are better than the out-of-the-box, default settings provided the database system. Referring back to the second SMART objective a concrete deliverable would thus be to create a novel workload mapping feature that can provide database settings that are at least 10% "better" than the out-of-the-box settings.

Another more challenging but equally interesting baseline could be to compare the database settings recommended by the original machine learning model versus the novel workload mapping model. Both baselines, so the out-of-the-box database settings as well as the settings recommended by the original machine learning model will be discussed in the results section.

3.4.2 Selected Database Performance Metric

There exist numerous metrics to evaluate whether the newly recommended database settings are "better" or not. The most popular ones as mentioned in [4, 48, 46] are throughput and latency. Both throughput and latency capture how fast a workload has been executed as further explained in the terminology section under database settings.

However, the main reason why throughput will be used is because it translates the optimisation problem into an $argmax$ problem. As such, the final settings that will be recommended to the user are the ones with the highest throughput based on the regression line. If latency was to be used then the optimisation problem would be an $argmin$ problem. Both approaches are valid but an $argmax$ problem seems to be conceptually more intuitive to work with as it follows the "the bigger, the better" principle. I.e. the bigger or higher the throughput, the better the overall database performance. The exact approach on how to solve this $argmax$ problem will be discussed in the design section.

4 Design

In the design section, the original database tuning architecture will be explained first. Then, the argmax detection problem and the sampling bottleneck problem will be examined in detail. Lastly, the novel similarity score, sampling function and ensemble model developed in this dissertation will be presented.

4.1 Original System

There exist many variations of different database tuning managers. Two of them (OtterTune and BTSTR) have been discussed in the background section. However, the majority database tuning managers roughly follow the same architecture [4, 46, 51, 24]. The entities of this general architecture, their interactions and the overall data flow will now be explained.

As rendered in figure 5 there exist three main entities in any general tuning architecture. Firstly, the database management system (DBMS) which is the entity meant to be tuned. Secondly, every database tuning architecture has a database benchmark. Without a database benchmark it is very difficult to assess the current performance level of the DBMS. Finally, every tuning architecture has a tuning manager which interacts with both the DBMS and the database benchmark.

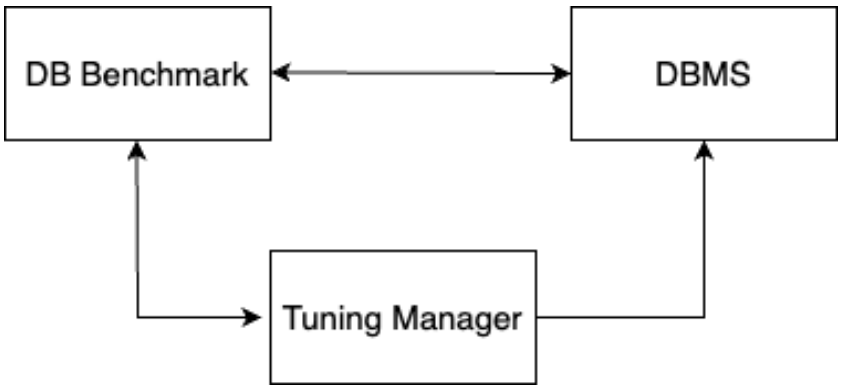


Figure 5: Bare-bones database tuning architecture

All entities interact bilaterally except for the tuning manager and the DBMS. Here the interaction only goes from the tuning manager to the DBMS but not the other way around. Reason being that any behaviour of the DBMS will only be channelled through the database benchmark entity.

To better understand the interaction between the entities it is easiest to first consider the tuning manager. One of the main responsibilities of the tuning manager entity is to update the database settings of the DBMS. The tuning manager entity also dictates which benchmark the database benchmark entity has to run against the DBMS. Finally, the tuning manager will use the data provided by the database benchmark entity to train a machine learning model such that the $argmax$ detection problem can be solved.

The database benchmark entity as previously explained runs benchmarks against the DBMS. The resulting performance values will then be sent to the tuning manager. Here it is important to note that one benchmark setting corresponds to one workload. I.e. the range of all possible benchmark settings that could be run against the DBMS correspond to the range of all possible workloads that could be tested in this database tuning architecture. More details about this will be provided in the implementation section.

As previously mentioned the DBMS entity is the entity meant to be tuned. Exact details on the specific settings that have been tuned will be provided in the implementation section. It is also important to note that there are many different database settings that can be manipulated by the tuning manager entity. Additionally, there exist out-of-the-box default values for each setting and the min and max values of said database settings depend the server's hardware profile. Again, more details will be provided in the implementation section.

Finally, when considering the data flow the key interaction to look at is between the database benchmark entity and the tuning manager. Data in form of "(database setting, database performance value) pairs" are collected by the database benchmark entity which are then forwarded to the tuning manager. Neither the DBMS nor the tuning manager actively generate new data.

4.2 Workflow

Before the exact approach to solving the $argmax$ detection problem can be elucidated the overall workflow of the previously introduced architecture needs to be further elaborated upon. As rendered in figure 6 said workflow has five steps in total which will now be explained.

First, it should be noted that figure 6 depicts the workflow for only one workload. However, the same workflow applies to any possible workload that can be generated. Thus, the first step of the tuning workflow is to choose a benchmark setting to emulated a specific workload. More details on how the database benchmark entity generates synthetic workloads will be provided in the implementation section.

The second step of the tuning workflow is the sampling process. More specifically it's the loop of updating the DBMS settings, running the benchmark which has been

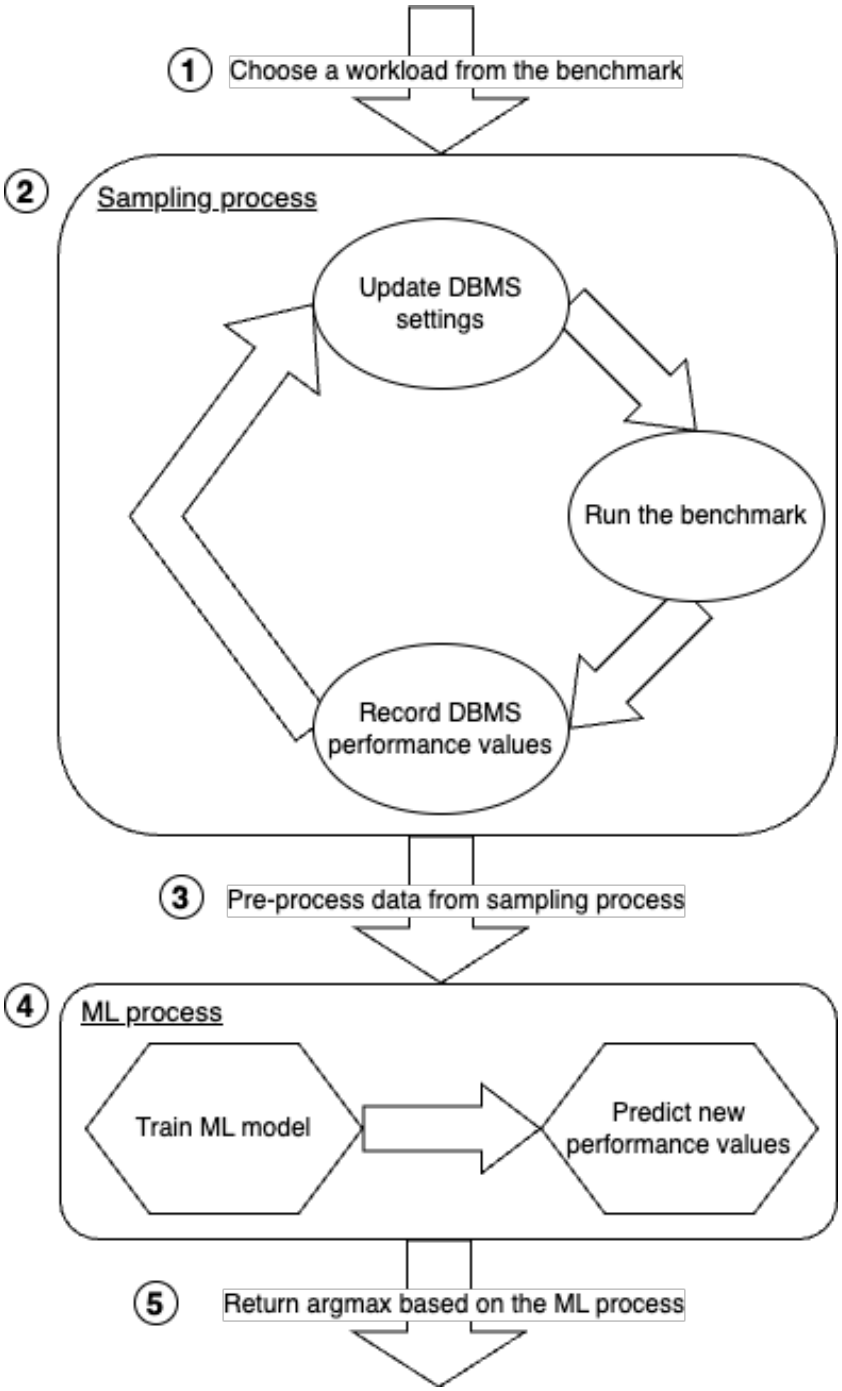


Figure 6: Workflow of the tuning architecture based on figure 5

specified in step one and then recording the DBMS performance values at the end of the benchmarking session. The entire sampling process is controlled by the tuning manager and allows it to collect data in the form of "(database setting, database performance value) pairs". The sampling process is an iterative process and the number of iterations controls the number of data points collected. So if the sampling process was to be run for $n = 100$ iterations than the tuning manager would have collected $n = 100$ "(database setting, database performance value) pairs". It is crucial to highlight again that these 100 data points correspond to one specific workload only which has been decided in step one.

The third step of the tuning workflow is to perform some data engineering on the samples which have just been collected in step two. This pre-processing step is crucial for the next step which is the machine learning (ML) process. Without any pre-processing no machine learning model can be adequately trained.

The ML processing step is the fourth step of the tuning workflow. Here, the data from step three is used to train a machine learning model. The goal is to fit a regression line such that database performance values can be predicted based on given database settings. As with most machine learning models the higher the number of training samples the better the model will learn. Once the model has been trained it will be used to make database performance predictions on unseen database setting values. This will be further explained in the default versus range settings section. As touched upon in the background section it is infeasible to benchmark all possible database settings combinations. Thus the goal of the ML model is to use a subset of database settings to then extrapolate the regression line over the range of all possible database setting values. Therefore, the test data set will be orders of magnitudes larger than the training set.

The final step of the tuning workflow is the *argmax* detection step. This step will also be carried out by the tuning manager and it uses the predictions made by the ML model from step four.

4.3 Argmax Detection

As previously mentioned the tuning manager uses the predictions made by the ML model for the *argmax* detection step. To best understand the *argmax* detection step consider figure 7 which serves as an example using mock data. The x-axis depicts the values 0 to 1 which correspond to the range of database settings available. As further explained in the default versus range settings section actual database settings have various ranges. The y-axis also depicts the values 0 to 1 and is denoted as $f(x)$ since the y-axis corresponds to the database performance values which are also a function of their database settings.

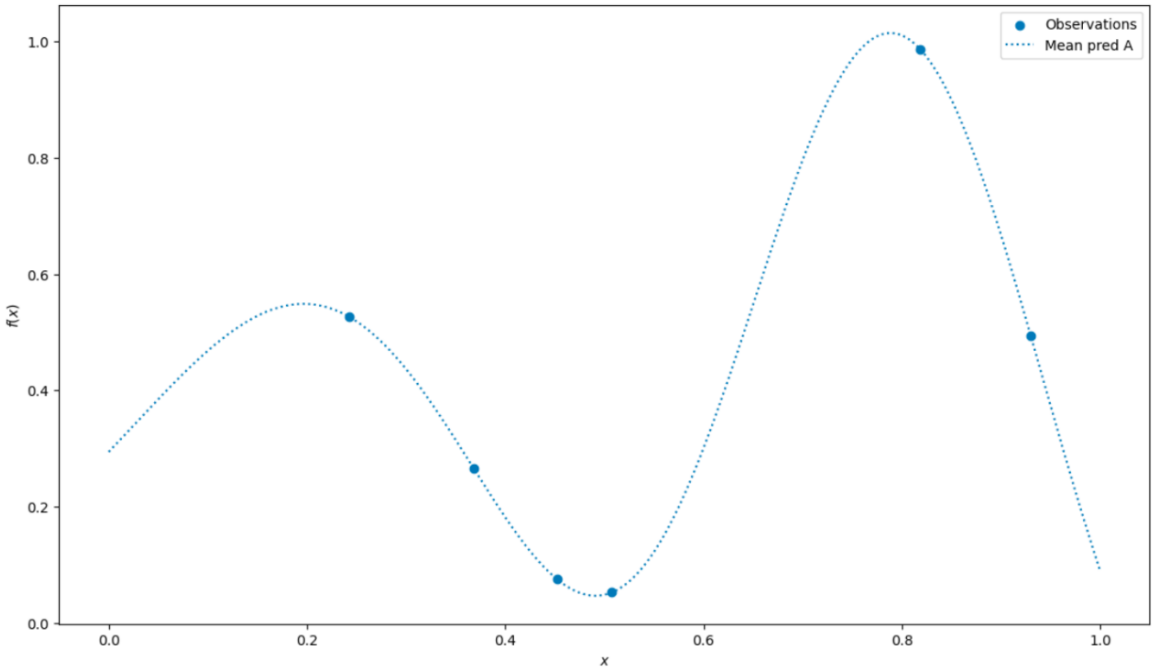


Figure 7: Argmax Detection

As explained in the workflow section the ML model tries to extrapolate the the shape of the entire function $f(x)$ based only on a subset of "(database setting, database performance value) pairs". In the example shown in figure 7 said pairs would be the $n = 6$ observations of shape $(x, f(x))$. Once the ML model has made a prediction for every x value between 0 and 1, it will return the most competitive database setting to the user which corresponds to the *argmax* of $f(x)$. In the example shown in figure 7 the *argmax* would be $x = 0.79$ so the most competitive database setting recommend to the user would be $x = 0.79$. In a real life tuning architecture x would be a list of database settings with specific *argmax* values per database setting category.

4.4 Sampling Bottleneck

As already mentioned at the very beginning in the motivation section the sampling process is the main bottleneck for most tuning managers when it comes to overall tuning time. As shown in figure 8 which is taken from the OtterTune paper [46] only the workload execution process (or benchmarking process as it is called in this dissertation) takes 500 seconds or about 8 minutes. Similar results have also been observed when implementing this dissertation.

When taking a closer look at figure 8 it shows the execution times of each workflow step in seconds for three different DBMS vendors, namely MySQL, Postgres and Vector. Vector is an OLAP database so it not of interest for this dissertation. However, the execution times of Postgres and MySQL are of interest. When considering the workflow presented in figure 6 then "workflow execution" translates to "run the benchmark and record performance values". "Prep & reload config" means "update DBMS settings". "Workload mapping" is not represented in figure 8 as it shows the workflow for a single workflow only. However, "config generation" corresponds to steps four and five, i.e. the ML and argmax process.

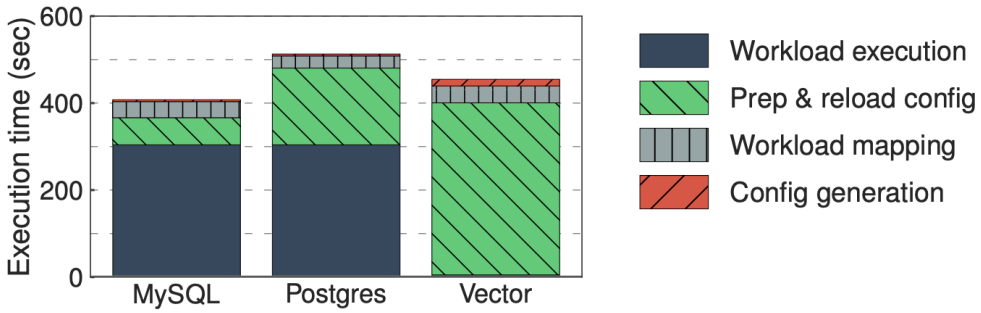


Figure 8: Sampling Bottleneck - Figure taken from [46]

It should be noted that the 500 seconds are a user defined parameter when analysing the sampling bottleneck problem. Thus, the running time can be shortened as it is user defined, however, this can lead to higher variance when recording the same benchmark for the same database settings. Especially, as performance metrics such as throughput are measured as "requests per second" so they are time bound. Therefore, there is little leeway when it comes to shortening the benchmark duration parameter.

As such, when considering the workflow from figure 6 it becomes clear that the iterative element of the sampling process (i.e. step two) is the main bottleneck. Once data has been collected training a ML model and returning the $argmax$ value (i.e. steps four and five) take very little time in comparison to the sampling process. However, since it is not really possible to reduce the benchmark duration parameter, the proposed solution in this dissertation is to **reduce the number of iterations of the sampling process to reduce the bottleneck.** Consequently, the ML model for the new workload might have less data points to work with which is a constraint that the novel workload mapping system needs to take into account.

4.5 Novel Workload Mapping System

The novel workload mapping system which will now be presented is the main contribution of this dissertation to the database tuning research landscape. As previously mentioned in the motivation section the intuition behind this novel approach is that if the new workload is very similar to the original workload, then the samples from the original workload can be re-used for the new workload. By re-using samples, less samples need to collected for the new workload and thus the sampling bottleneck is reduced.

4.5.1 Similarity Scores

Before introducing the novel sampling function the similarity score on which said sampling function is based upon needs to be explained. Other tuning managers such as OtterTune [46] or BTSTR [4] use distance based similarity metrics such as the Euclidean distance or vector based similarity metrics like the cosine similarity score.

The main advantage of the Euclidean distance measure is that is is easy to compute and interpret. However, the Euclidean distance suffers from the "curse of dimensionality", i.e. it does not apply well to high dimensional data [11]. This is where the cosine similarity has an advantage over the Euclidean distance measure as it just measures the angle between two feature vectors regardless of their size or dimensionality [13]. Both approaches work well when comparing the $argmax$ database settings returned for W_o and W_n. Reason being that returned database settings are a list of $argmax$ values for each database setting category so the lists can be used as vectors.

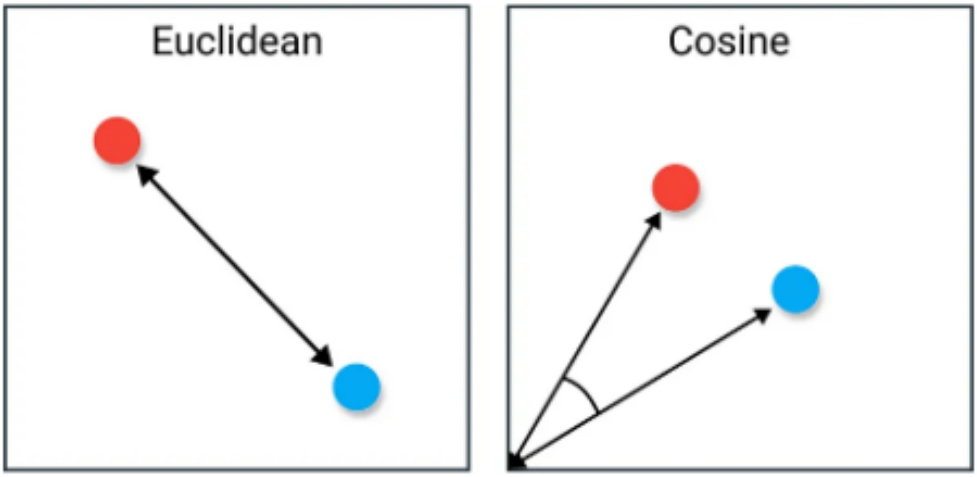


Figure 9: Euclidean and Cosine Similarity [13]

However, the goal of this new similarity score is to gain an idea of how similar W_o and W_n are before sampling. Otherwise, the number of samples that need to be collected for the new workload W_n is equal to the number of samples collected for the original workload W_o. Equal number of samples means that the sampling bottleneck issue has not been addressed. Thus, $\hat{\theta}$ is of interest which is derived from w_o and w_n, i.e. before any sampling has occurred.

As outlined in more detail in the implementation section it is possible to compute a pre-sampling similarity score $\hat{\theta}$ between w_o and w_n based on the transaction logs of the benchmark. More specifically when analysing the default settings or out-of-the-box database settings in more detail. In the context of the benchmark the log is just a file containing every single transaction executed against the database and its duration. Exact implementation details will be discussed later. However, by comparing the logs (for the default values) of the original versus the new workload, the number of samples to be collected for the new workload can be determined.

4.5.2 Jensen Shannon Divergence

To compare the transaction logs of the default values the Euclidean distance measure cannot be used since the data is too high dimensional. However, in order to use the cosine similarity score both vectors for w_0 and w_n must contain the same number of elements. This is not the case for the un-processed log data since different default settings will result in different throughput values. Since throughput is measured in "requests per second" the total number of transactions executed within a pre-defined timer interval

(e.g a 500 second window) will be different for w_0 and w_n. Consequently, the log data needs to processed such that the total number of elements is the same.

One possible approach to ensure that w_0 and w_n have the same number of elements is by transforming the logs of w_0 and w_n into two histograms with the same number of bins. However, now the log data describes the underlying distribution of w_0 and w_n as well as having the same number of elements. This additional information can be leveraged by using distribution based similarity metrics such as the Jensen Shannon divergence score [28]. The cosine similarity score would still work, however, it would be ignoring the distribution of the data set at hand.

$$\begin{aligned} D_{JS}(P|Q) = D_{JS}(Q|P) = \frac{1}{2}D_{KL}(P|M) + \frac{1}{2}D_{KL}(Q|M) \\ M = \frac{1}{2}(P+Q) \end{aligned} \tag{5}$$

The Jensen Shannon divergence (JSD) score as shown in equation 5 is a statistical distance measure that allows for the comparison between two probability distributions P and Q. Here P represents the old workload w_o (before sampling) and Q stands for the new workload w_n (also before sampling). It is noteworthy that JSD is the "bounded symmetrization of the unbounded Kullback–Leibler divergence" as explained by Nielsen in 2019 [31]. As such, one of its major advantages is that ordering does not matter by symmetry, i.e. $D_{JS}(P|Q) = D_{JS}(Q|P)$ have the same score. In order to achieve said symmetry the average of w_o and w_n must be computed which is denoted as M in equation 5.

Additionally, if base 2 is chosen for the logarithm for the Kullback–Leibler divergence as shown in equation 6 then the JSD score is a real number bounded between 0 and 1. A JSD score of 0 means that the two distributions P and Q are identical. 1 means that the two distributions are completely different.

To put it in a nutshell, **JSD is the ideal similarity score to compare w_o and w_n because** it is a statistical distance measure, it is symmetric and it is bounded between 0 and 1 meaning that the similarity can be expressed as a percentage. Additionally, the JSD score has not been used in the database tuning research landscape in the context of reducing the sampling bottleneck. Thus this is a novel approach and an active

contribution to the field. The only paper which also uses the JSD score in the context of DB tuning is by Aslam et al. from 2007 [3] where the authors use the JS divergence estimate the difficulty of queries which is a different focus than this dissertation.

$$D_{KL}(P|M) = \sum_{x \in X} P(x) \log_2 \left(\frac{P(x)}{M(x)} \right) \tag{6}$$

To conclude this section the Kullback–Leibler divergence (KLD) as shown in equation 6 needs to be explained. As previously mentioned JSD is just a transformation of the KLD so the actual similarity measure is calculated by the KL divergence which is also known as relative entropy [7]. It is also worth mentioning that capital X in equation 6 is the set of all bins that have been created when transforming the log data into histograms. More information can be found in the implementation section.

4.5.3 Sampling Function

The sampling function which is based on the JS divergence is the main contribution of this dissertation to the database tuning research community. It is novel, light-weight and can easily be understood and implemented as presented in equation 7.

The intuition behind this function is the same as the intuition presented at the very beginning of this dissertation in the motivation section. If two workloads are similar then samples ought to be re-used. If not then new samples have to be collected as no model could be trained otherwise. As such, the sampling function has an upper and lower bound which is captured in the sampling function.

$$n_n = f(n_o, \hat{\theta}) = \hat{\theta} * n_o \qquad ; \quad \text{and} \quad 0 \leq \hat{\theta} \leq 1, \quad n_o, n_n \in \mathbb{N} \tag{7}$$

In equation 7 n_n denotes the number of samples for the new workload. Conveniently, the JS divergence quantifies the similarity between w_o and w_n as a percentage between 0 and 100%. As such, the number of samples which ought to be collected can be calculated as a function of the JS divergence and the number of original samples collected. If w_o and w_n are completely different then the JSD score is 1. Thus the number of samples which should be collected for the new workload w_n need to be the same as n_o (i.e. 100% times n_0) which is the original number of samples collected. On the contrary, if w_o

and w_n are the same then the JSD score is 0, meaning that 0 new samples need to be collected.

As such, the sampling bottleneck can be drastically reduced by minimizing the number of samples which ought to be collected for the new workload w_n. However, sampling is only one step in the database tuning workflow and the ML process (as shown in step two of figure 6) needs to be updated accordingly such that database settings returned to the user remain competitive.

4.5.4 Ensemble Model

The following ensemble model presented in equation 8 is the natural progression of the ML process shown in figure 6 based on the new sampling function which has been introduced in equation 5.

$$y_n = \begin{cases} M_o(W_o) & ; \hat{\theta} = 0 \\ M_n(W_n) & ; \hat{\theta} = 1 \\ M_o(W_o)(1-\hat{\theta}) + M_n(W_n)\hat{\theta} & ; 0 < \hat{\theta} < 1 \end{cases} \tag{8}$$

An ensemble model or ensemble modelling as explained by Kotu et al. in 2015 [23] is "a process where multiple diverse models are created to predict an outcome, either by using many different modeling algorithms or using different training data sets". Without this new ensemble model the original workflow would be disregarding historical models resulting in inadequate database settings to be returned.

When taking a closer look at equation 8 it becomes clear that it respects the same bounds as the sampling function presented in equation 7. The main difference is that the equation returns a database performance prediction value y_n for the new workload as opposed to a number denoting the quantity of new samples which ought to be collected. Therefore, y_n is a function of $M_o(W_o)$ and $M_n(W_n)$ where M denotes a machine learning model. The remaining notation is the same as summarised in table 1.

As previously mentioned the two boundaries are the same as for the sampling function. Hence, when the new workload is completely different compared to the original one then all samples should be used to train a new model $M_n(W_n)$. Similarly, if all samples can

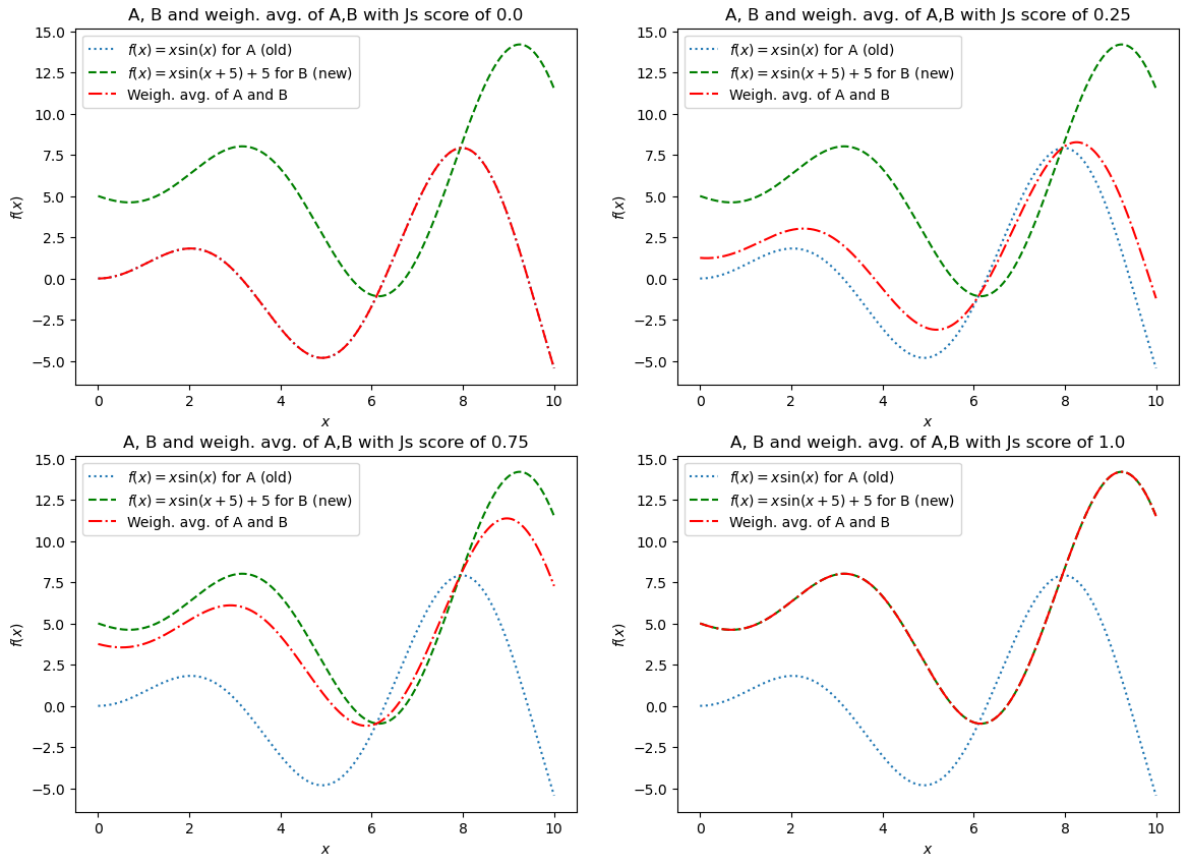


Figure 10: Ensemble model based on equation 8

be re-used then the old model $M_o(W_o)$ needs to be re-used. Thus, the ensemble model in equation 8 is designed in such a way that it respects both boundaries if the JSD score is between 0 and 1 by combining both models $M_o(W_o)$ and $M_n(W_n)$ in a weighted manner. When designing this model various discrete probability distributions such the Binomial or the Uniform distribution have served as inspiration.

Finally, the ensemble model presented in equation 8 can be graphically represented as rendered in figure 10. There exist three graphs in each sub-figure. Blue and green denote the regression lines for workloads A and B and red is the ensemble model based on the regression lines for A and B.

When considering the bottom right sub-figure in figure 10 it shows the red ensemble line on top of the green regression line for workload B. Reason being that the JSD score is 1 meaning that 100% of the new samples should be used for workload B. In other words, the old workload A implies no information at all about the new workload B so workload A needs to be forgotten.

Similarly, the top left sub-figure shows the red ensemble model line when the JSD score is 0. In this case all samples of the original workload A can be re-used, meaning that the red ensemble model line will be the same as the regression line for workload A.

As expected, the red regression line for a JSD score of 0.25 and 0.75 is plotted between the bounds imposed by the blue line for workload A and the green line for workload B. If the JSD score was to be 0.5 then the red regression lines would be precisely in the middle between the two bounds. This means that the JSD score dictates the "trust ratio" between the original and new model $M_o(W_o)$ and $M_n(W_n)$. The higher the JSD score the more the new model will be trusted as it has more sample points to train with since the new workload is less similar compared to the original workload.

5 Implementation

In the following sections, the implementation details based on the previously introduced design decisions will be elucidated. Additionally, the difference between default and min, max values will be clearly explained. When reading the following sections it is important to keep in mind that no formal software engineering requirements have been defined as this is a research-based dissertation. Thus, all implementation details are proof-of-concept realisations of the ideas presented in previous chapters.

5.1 Selected Database Benchmark

The selected database benchmark for this dissertation is the TPC-C [41] benchmark. TPC stands for transaction processing performance council and the version C benchmark is an OLTP benchmark. As mentioned in the terminology section a workload is a specific use case for which samples are collected. Thus, a benchmark is needed to generate workloads and collect their corresponding database performance values. To this end, the TPC-C benchmark is used which populates the database with mock data, runs synthetic workloads against the database and finally collects the database's run-time statistics.

However, as argued by Zhang from Brown University in 2018 [50] the TPC-E [42] benchmark has now replaced the TPC-C benchmark as the de facto industry standard. Thus, the TPC-E benchmark was initially considered for this project. However, one of the major problems with any TPC benchmark is the actual setup on a local machine. Even though TPC publishes all information about the architecture of their benchmarks, there exists only one open source version supported on the official TPC GitHub repository. Said version is called HammerDB [40] and is based on the TPC-C benchmark. However, upon further research no TPC-E implementation could be found online. As such TPC-C was chosen and deemed as appropriate as is still used in the database tuning literature [8] today.

5.2 OLTP Bench

OLTP Bench is an open source implementation of the TPC-C benchmark and can be found on GitHub [2]. It has been developed as a research project by Pavlo et al. since

2014 [8] who also wrote the OtterTune paper. OLTP Bench can be run from the command line as further explained in the implementation section and also provides a detailed overview of the recorded performance values. Due to the ease of running OLTP Bench and its popularity in the research literature [4, 46, 45] it was chosen as the benchmarking tool for this dissertation.

5.2.1 Tables

The TPC-C schema as shown in figure 11 has nine tables in total. The benchmark is meant to model a wholesale supplier with multiple, geographically separated sales districts. Each district is meant to serve 3,000 customers with each warehouse storing 100,000 items. Based on the OLTP Bench set-up file (as shown in figure 12) the number of warehouses can be controlled. The number of warehouses then indirectly controls the amount of space the database will take up on the server on which the DBMS runs.

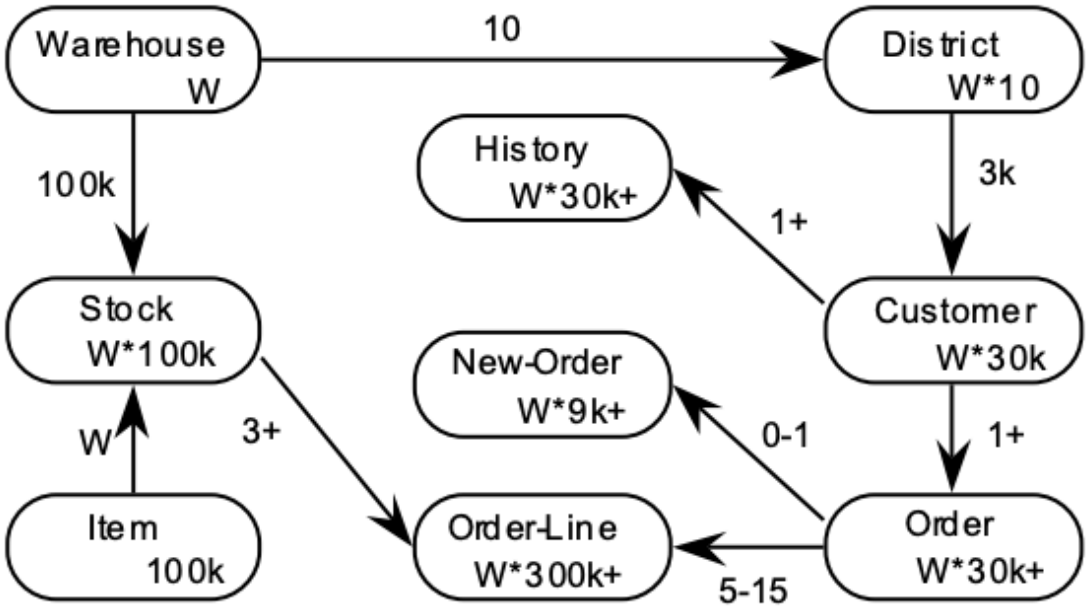


Figure 11: TPC-C Schema [41]

5.2.2 Workload Generation

As previously mentioned, OLTP Bench allows the user to control precisely which workload to generate and how to execute it against the database. The workload generation mechanics will now be explained.

```
<?xml version="1.0"?>
<parameters>

    <!-- Connection details -->
    <dbtype>postgres</dbtype>
    <driver>org.postgresql.Driver</driver>
    <DBUrl>jdbc:postgresql://localhost:5432/postgres</DBUrl>
    <username>valentinkodderitzsch</username>
    <password>tpcc</password>
    <isolation>TRANSACTION_READ_COMMITTED</isolation>

    <!-- Scale factor is the number of warehouses in TPCC -->
    <scalefactor>10</scalefactor>

    <!-- The workload -->
    <terminals>10</terminals>
    <works>
        <work>
          <time>300</time>
          <rate>2000</rate>
          <ratelimited bench="tpcc">true</ratelimited>
                  <weights>45,43,4,4,4</weights>  <!--  A 10-->
                  <!-- <weights>45,4,4,4,43</weights>  B 10-->
                  <!-- <weights>5,45,40,5,5</weights>   C 10-->
                  <!-- <weights>30,45,2,20,3</weights>   D 10-->
                  <!-- <weights>30,45,5,10,10</weights>   E 10-->
        </work>
        </works>

        <!-- TPCC specific -->
           <transactiontypes>
            <transactiontype>
                    <name>NewOrder</name>
            </transactiontype>
            <transactiontype>
                    <name>Payment</name>
            </transactiontype>
            <transactiontype>
                    <name>OrderStatus</name>
            </transactiontype>
            <transactiontype>
                    <name>Delivery</name>
            </transactiontype>
            <transactiontype>
                    <name>StockLevel</name>
            </transactiontype>
           </transactiontypes>
</parameters>
```

Figure 12: OLTP Set-Up File [2]

When considering the OLTP Bench set-up file depicted in figure 12, it becomes evident that there exist four main parameters. The first parameter is the connection details. This parameter assumes that the corresponding DBMS is up and running and at the start of each benchmarking session OLTP Bench will try to connect to the DBMS using the provided credentials.

The second parameter is the scale factor. As previously mentioned, this parameter indirectly controls the total size of the database. The bigger the scale factor the more secondary memory the DBMS will use on the server.

The third parameter is the most important parameter as it controls how the workload is meant to be generated. In the context of reducing the sampling bottleneck, the time parameter is most important as it determines the total running time of the benchmark in seconds.

The second most important parameter in the context of workload generation is the weights parameter. This parameter determines the unique workload executed against the database. Each weight represents a TPC-C specific transaction type. The sum of all weights must always be 100. As such, the total number of all possible workloads which can be generated is about 4.5 million. This number has been determined by using the stars and bars combinatorics approach [10].

The final parameters to consider are the TPC-C specific transaction types. There exist five unique transaction types in total which execute both read-and-write operations against the database. As such, the five transaction types manipulate the size of the database and perform smaller analytical queries. It is important to note that every workload is a unique combination of these five transaction types. The exact combination for each workload is defined by the user via the weights parameter.

5.3 Selected Database Settings

There exist a plethora of database settings to choose from as explained in the background section. Consequently, some end-to-end tuning managers such as OtterTune incorporate dimensionality reduction techniques such as factor analysis in their tuning pipeline. However, it is well known in the database tuning community that there is a list of

approximately 15 (out of hundreds of settings) that seem to be the most relevant for performance tuning [37]. Said list has been empirically verified in papers such as [46, 48, 51]. Therefore, the work presented in this dissertation will be based on said list.

It is important to note though that these empirically validated settings are linked to specific DBMS vendors such as MySQL or PostgreSQL. Both MySQL and PostgreSQL are two of the most popular open-source DBMS solutions. They are both relatively easy to set up and OLTP Bench is compatible with both of them. Additionally, a list of pre-selected, relevant knobs exists for both MySQL and PostgreSQL. Thus, there are very few quantitative differences between the two. However, in the end, PostgreSQL version 9.6 was chosen for this dissertation as the code base provided by Barbulescu uses PostgreSQL version 9.6. This decision helped speed up the development process of the novel workload mapping feature as the PostgreSQL DBMS was already integrated into the tuning manager.

The final list of chosen database settings is shown in table 3 and is based on the work of BTSTR [4] and OtterTune [46]. The explanations for each setting are taken from the PostgreSQL wiki page [18]. This means that the database settings presented in table 3 are the independent variables. Hence, any recorded database performance values will be based on the settings presented in table 3.

Table 3: Selected PostgreSQL settings

Setting	**Type**	**Description**
effective_cache_size	RAM	Sets the amount disk cache available for a single query
maintenance_work_mem	RAM	Sets the amount of memory to be used by maintenance operations (e.g. Create Index)
max_wal_size	Storage (log)	Sets the write ahead log (WAL) size that triggers a checkpoint
max_worker_processes	CPU	Sets the maximum number of concurrent worker processes
shared_buffers	RAM	Sets the size used for shared memory buffers
temp_buffers	RAM	Sets the maximum number of temporary buffers used by each session
wal_buffers	RAM	Sets the amount of shared memory of WAL data that has not been flushed yet
work_mem	RAM	Sets the amount of memory for internal sorting and hash operations

5.3.1 Default vs Range of Settings

As previously mentioned, the range of possible database settings depends on the hardware profile of the machine used to host the DBMS. For the purpose of this dissertation, the majority of the work was done on an M1 Macbook since students do not have sudo permissions on the DCS machines. However, to set up the database tuning architecture a legacy version of PostgreSQL needs to be deployed. Additionally, the tuning manager needs to execute flushing commands in the terminal when restarting the database which also requires sudo permissions. Therefore, it was concluded at the time that the development process can be sped up if the majority of work was completed on a personal laptop.

Name	**Min**	**Default**	**Max**
effective_cache_size	8 KB	4 GB	80% of 8GB
maintenance_work_mem	1 MB	64 MB	80% of 8GB
max_wal_size	32 MB	1 GB	80% of 5GB
max_worker_processes	0	4	8
shared_buffers	128 KB	8 MB	80% of 8GB
temp_buffers	128 KB	8 MB	80% of 8GB
wal_buffers	64 KB	64 KB	80% of 8GB
work_mem	64 KB	4 MB	80% of 8GB

It is crucial to understand the difference between the default and the min, max values as they are the difference between the pre-sample workload w and the post-sample workload W. The default values are the out-of-the-box configurations provided by PostgreSQL. The main idea is to leverage the logs provided by OLTP Bench for the default values to compute a similarity score $\hat{\theta}$ between the pre-sampling workloads w_o and w_n. This can be done very easily, as there is only one default value per database setting type.

However, the post-sampling workload W requires a more costly sampling process. The min and max values provide the range for the Latin Hypercube Sampling (LHS) technique and $n = 100$ samples will be collected. This sampling process for $n = 100$ will take approximately 10 hours as further discussed in the results section.

The number of samples n is 100 for the original workload W_o since there are eight database settings and any regression model needs at least 10 samples per feature dimension. As such, the lower bound is $n = 80$, however, $n = 100$ is chosen to ensure better

predictions for W_o. As mentioned in previous sections the main assumption here is that the pre-sampling similarity score $\hat{\theta}$ (which is based on the logs of the out-of-the-box settings) is the same as the post-sampling similarity score θ (which is based on $n = 100$ samples).

5.4 Novel Workload Mapping Feature

So far the difference between min, max values and the default values have been explained. Now, the actual process of computing the pre-sampling similarity score $\hat{\theta}$ will outlined in detail.

5.4.1 Transaction Logs and Similarity Score

As shown in table 4 the logs record the duration of every single transaction type within the user-defined time interval of the OLTP Bench benchmark. Said log file is saved as a .csv file and can thus be easily imported into a Jupyter Notebook [21] for analytic purposes.

To this end, the following python libraries have been used: Numpy [32], Pandas [34], Scipy [38], Sklearn [39] and Matplotlib [39]. Firstly, the logs are analysed using numpy to determine the "weights" or frequency of each transaction type based only on the data from the logs. Then, the log data for each transaction type is turned into a histogram with $n = 1000$ bins (as shown in figure 15 in the results section). The five histograms and their according "weights" now describe the original pre-sampling workload w_o. A graphical representation of this can be found in the results section.

Then, the same process of using the default values is repeated for the new workload w_n. Again, the new workload w_n can be described using the five histograms from the transaction logs and their "weights". Now, the JSD score can be determined for each transaction type from w_o and w_n. To this end, the JSD implementation of the scipy library will be used. Thus, there will be **five JSD values between w_o and w_n**. One JSD value for each transaction type. An overall similarity score $\hat{\theta}$ will be returned by computing a weighted average of the previously mentioned five JSD values. The weights will be the same as the "weights" calculated from the log data of the new workload.

Table 4: Log Data for Default Settings (Top 10 logs)

Transaction Type Index	**Transaction Name**	**Start Time (microseconds)**	**Latency (microseconds)**	**Worker Id (start number)**
2	Payment	1676992765.368736	45407	2
5	StockLevel	1676992765.368963	92600	5
5	StockLevel	1676992765.368966	86165	6
1	NewOrder	1676992765.368975	62919	7
1	NewOrder	1676992765.370587	61067	3
1	NewOrder	1676992765.370593	62917	0
5	StockLevel	1676992765.370617	88214	1
5	StockLevel	1676992765.371860	85889	9
4	Delivery	1676992765.371862	84069	8

5.4.2 Sampling Function and Ensemble Model

At this point, the similarity score $\hat{\theta}$ has been determined based on the information provided by the log data of the default values. Now, the number of samples for the new workload can be easily determined by plugging in the numbers into equation 7 for parameters $\hat{\theta}$ and n_o. Here, n_o represents the number of samples for the original workload. Depending on the similarity score $\hat{\theta}$, the sampling bottleneck might be dramatically reduced if the original and new workload are fairly similar.

Once the sampling function has determined the number of samples to be collected for the new workload, n_n samples will be collected by running the OLTP Bench benchmark n_n times. Here it is important to point out again that for each iteration the tuning manager will load a different database setting. Said settings will have been determined by applying the LHS technique for n_n points over the min, max range of all possible database settings (which is hardware specific).

After n_n samples have been collected for the new workload, a model M_n can be trained using said samples. The final prediction will be returned by the ensemble model as described in equation 8 which is graphically represented in figure 10. To this end, a random forest tree and a Gaussian regression model have been used. Both of these have been implemented using sklearn.

Concrete tuning results will be discussed in the results section. However, it is important to highlight that this implementation will only make ONE prediction per workload since this approach is not meant to be part of an end-to-end tuning manager. Thus, implementing this similarity score and ensemble model into an end-to-end tuning solution is reserved for further work.

5.5 Experimental Setup

An exact guide on how to recreate the results achieved in this dissertation will be provided in the Readme file of the code submission. However, the most important recreation steps are:

1. Run the code on a MacOS operating system (Ventura 13 or higher)
2. Install PostgreSQL version 9.6

3. Install OLTP Bench

4. Set up PostgreSQL and OLTP Bench as defined in figure 12

6 Results

In the results section, the experimental findings of the sampling bottleneck problem will be discussed first. Subsequently, the results of the Jensen-Shannon divergence as implemented in this dissertation will be discussed. Finally, the ideas presented in the design section will be evaluated against the baselines defined in the methodology section. In the end, the results show that the initially defined research objectives are met.

6.1 Sampling Bottleneck

As mentioned numerous times in this dissertation the sampling process is one of the main bottlenecks in most database tuning architectures. Figure 13 shows the execution times per workflow process for a single data point as observed in this dissertation.

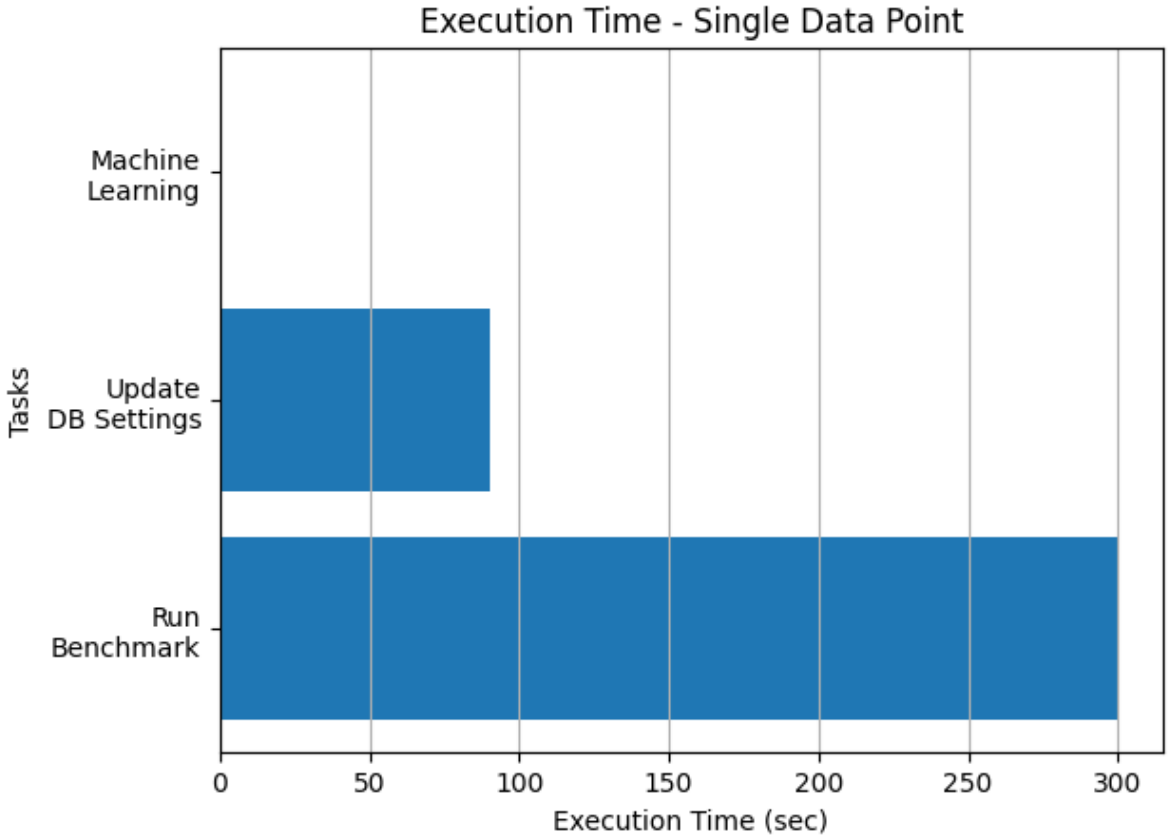


Figure 13: Duration per Workflow Process (Single Data Point)

As such, figure 13 is empirical proof for the observed sampling bottleneck when running the tuning architecture provided by Barbulescu. Most notably, the "run benchmark" task takes 300 seconds for a single data point as it is a user defined parameter. Updating the database settings takes about 90 seconds per data point as some database

setting categories require a database restart in order to become active. Running the machine learning task requires zero seconds per data point though as the ML step is only performed at the end of the entire sampling process.

Since collecting a single data point took at least 390 seconds only 100 data points have been collected per workload for this dissertation. Because of the small data set size of $n = 100$ pre-processing the samples and training the ML model is completed almost instantaneously in a matter of seconds. Whereas sampling $n = 100$ data points takes about 11 hours. This discrepancy between the sampling time and then the model training time is also meant to be hinted at in figure 13 by intentionally leaving the bar for the ML task empty. Reason being that 45 seconds cannot be visualised on the same plot if the x-axis has a range between zero and 11 hours.

It is also worth mentioning that figure 13 is reminiscent of figure 8 presented in the design section. Thus, figure 13 shows that the observations made by Van Aken et al. in [46, 45] have been reproduced in this dissertation when analyzing the sampling bottleneck.

6.2 Jensen-Shannon Divergence

As shown in the following two sections the JSD score has successfully been implemented when comparing single transaction types as well as entire workloads. As the proposed similarity metric for this dissertation has been successfully computed the novel sampling function and ensemble model can also be successfully implemented.

6.2.1 Single Transaction Type

Figures 14 and 15 show two histograms for the "new order" transaction type based on workloads A and B. "New order" is one of five transaction types constituting a workload as defined by the OLTP Bench benchmark [8]. Both histograms can be used to compute a JSD score between workloads A and B but only for the "new order" transaction type. The two histograms are best understood when first considering figure 14.

As shown in figure 14 the top and bottom histograms have 6 bins each. It is important to recall that the log file record how many times and for how long each of the five transaction types have been executed within the pre-defined time window. As such,

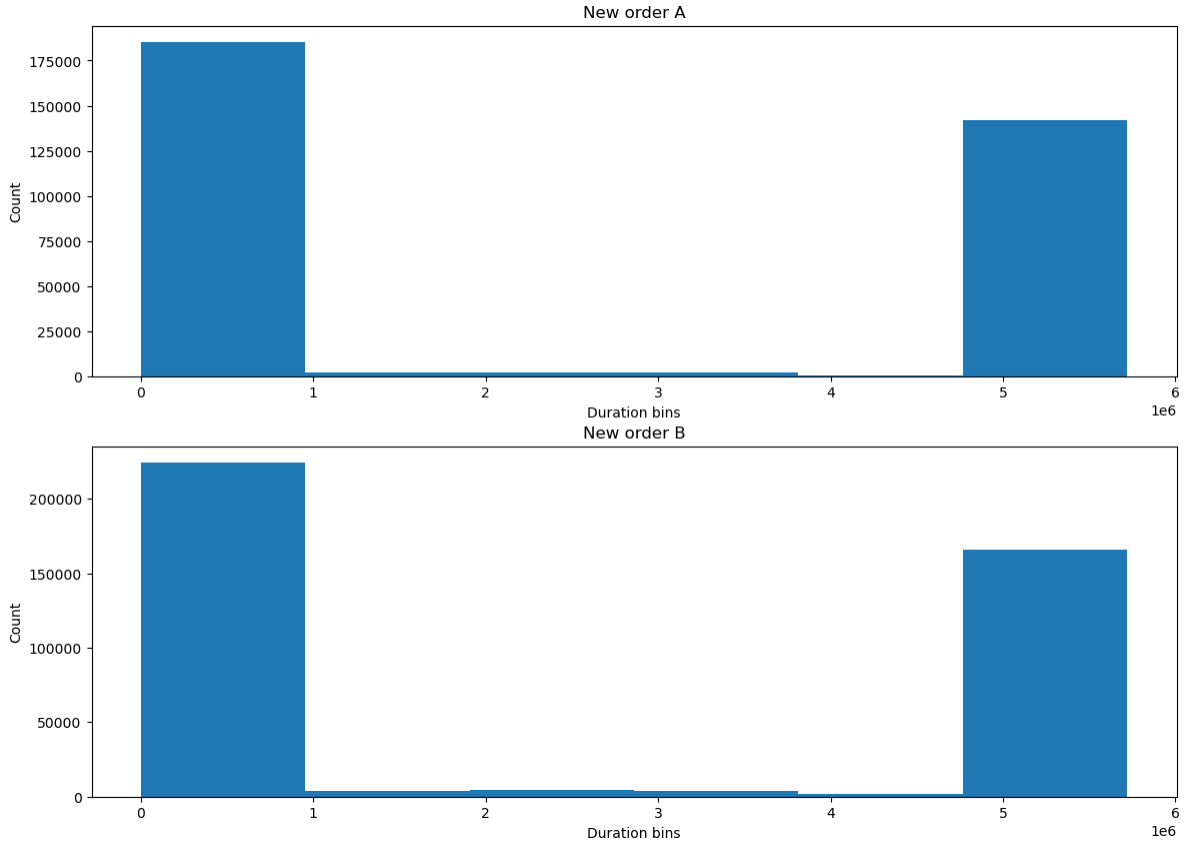


Figure 14: Histogram for "New Order" transaction type for workload A vs. B - JSD score of 0.0447

the histogram tells the reader that about 175 thousand transactions have been recorded in the log file with a duration of under one second each. The log file measures the transaction duration in microseconds and the x-axis of the histogram has a scale of 10^6 so each tick denotes one second.

When further analysing figure 14 it stands out that there are very few (almost zero) transactions recorded with a duration between one and 4.7 seconds. However, there are about 150 thousand transactions recorded with a duration between 4.7 and 5.7 seconds. As such, figure 14 shows a bimodal distribution which can be interpreted as an indicator that the OLTP Bench benchmark [8] is indeed a synthetic workload generator.

As already mentioned in the design and implementation sections the JSD score can be computed based on the histogram presented in figure 14. However, since only six bins have been used for figure 14 the histograms for workloads A and B are very "low resolution" resulting in a JSD score of 0.0447 indicating that the two distributions are nearly identical.

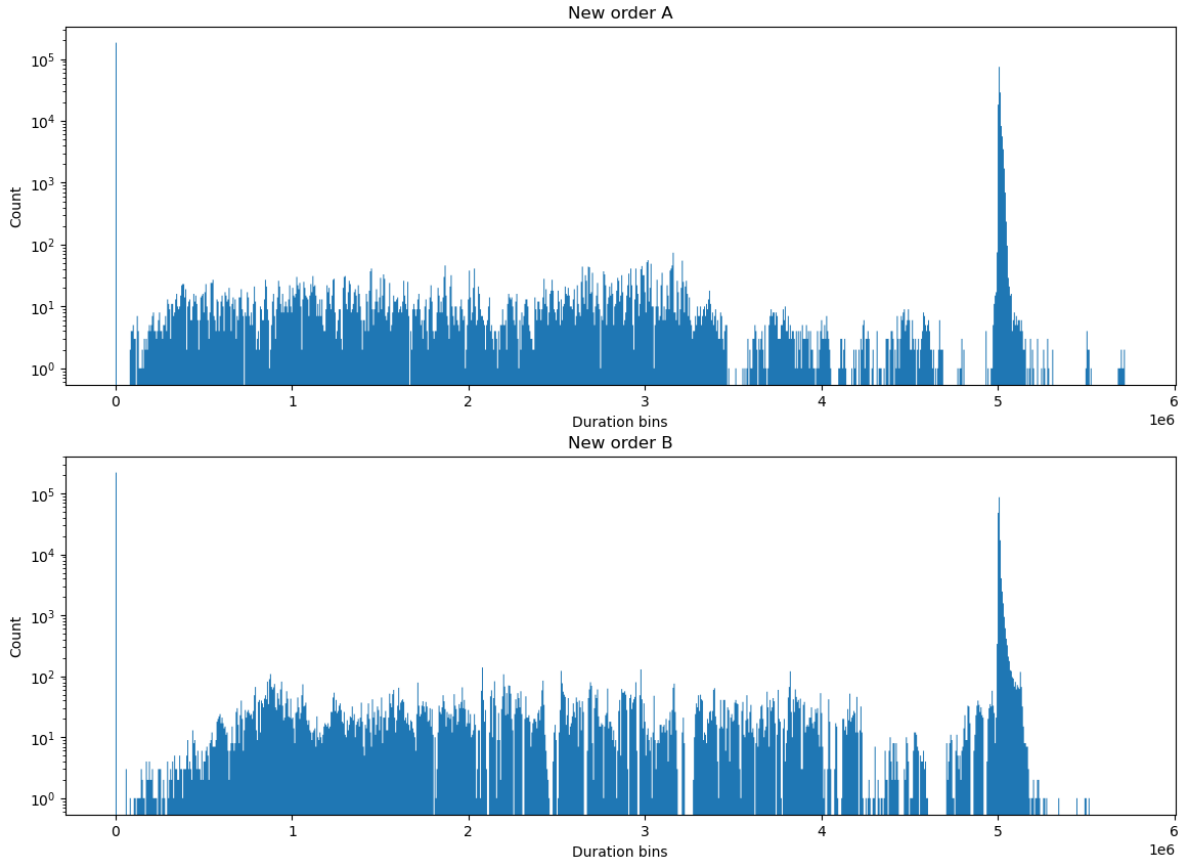


Figure 15: Histogram for "New Order" transaction type for workload A vs. B - JSD score of 0.1768

However, when increasing the "resolution" of the histograms from six to 1000 bins as shown in figure 15 then the JS divergence is able to capture more differences. As such the JSD score increases from 0.0447 to 0.1768 indicating that the workloads A and B for the "new order" transaction types are still very similar but not almost identical.

It is also important to realize that the y-axis of figure 15 is on a log-scale resulting in the transactions between one and 4.7 seconds to be "blown up" in the visualization. However, for the purpose of computing the JSD score this has no implication as the log-scale is just used for visualization purposes. Without the log-scale the "valley" between the two peaks would be visualized as non-existent which is not accurate.

In conclusion, figures 14 and 15 show that the JSD score can be successfully calculated for a single transaction type based on the log files of the default values. As such it is possible to determine the similarity between two workloads by only using the logs of the out-of-the-box database settings. This is a desirable result as it allows for further development of an overall JSD score. Said JSD score can be used to compute the similarity

score $\hat{\theta}$ for entire workloads as opposed to just a single transaction type. Consequently, this result is a step in the right direction to reducing the sampling bottleneck.

6.2.2 All Transaction Types

Based on the results for the JSD score when comparing single transaction types it is possible to compute an overall similarity score $\hat{\theta}$ by computing a weighted average of all five transaction types. The exact details have been discussed in the implementation section.

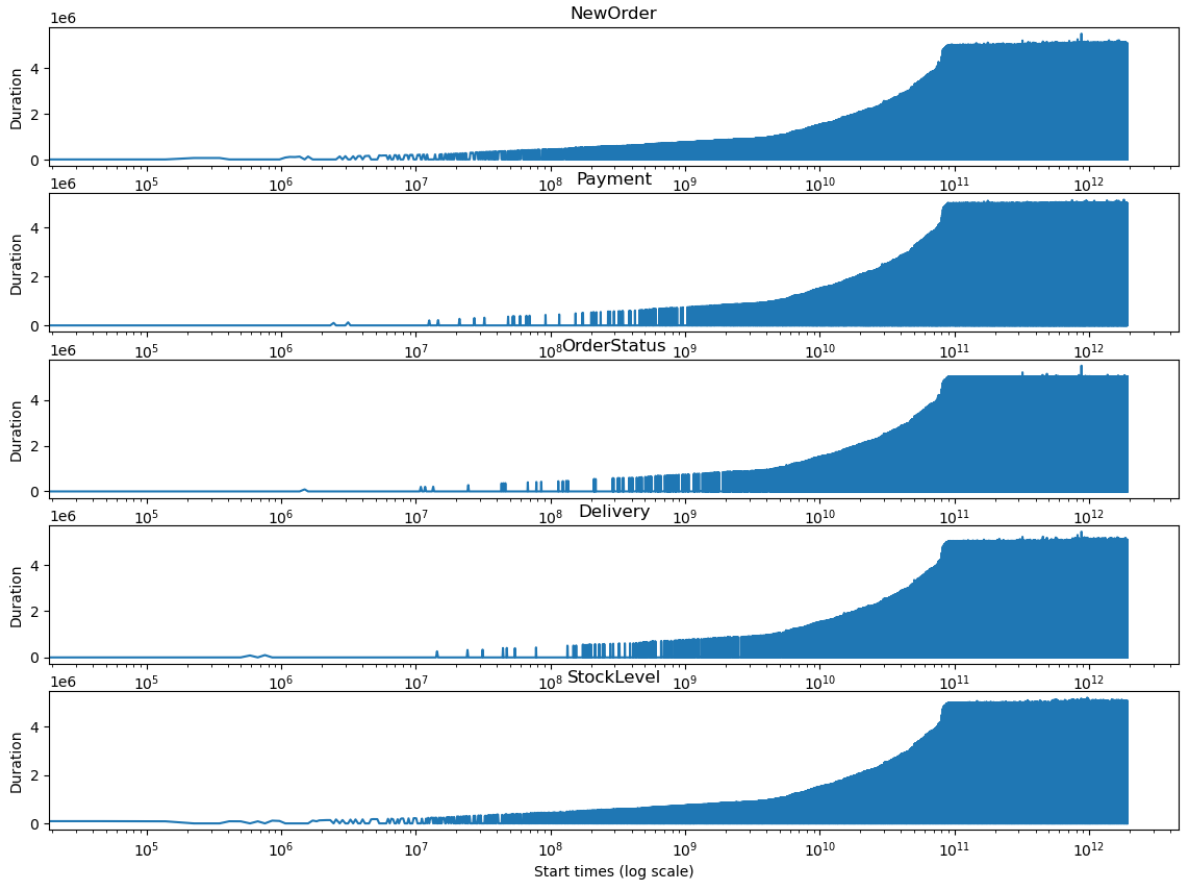


Figure 16: Duration per transaction type

Figure 16 shows the duration of each transaction type per workload. Again, a workload is synthetically generated by the OLTP Bench benchmark by interleaving five individual transaction types. In figure 16 their duration are shown on the y-axis where each tick denotes 2 seconds. As opposed to the histograms from figures 14 and 15 this graph simply plots the duration of each transaction in chronological order. As such, it becomes evident that the duration of each transaction type increases as time passes by. This was expected since some transactions change the size of the database resulting in longer

query processing times.

This is an insightful result as it shows that it is possible to analyse all five transaction types at once. As such, it is possible to compute an overall similarity score $\hat{\theta}$ based on the transaction logs of the out-of-the-box database settings alone. Consequently, this result can be interpreted as a successful proof of concept that the JS divergence can be applied to map between workloads. As such, the JS divergence can be seen as a more versatile tool for database tuning in general than initially proposed by Aslam et al. in [3] for query hardness estimation alone.

6.3 Baselines

So far in this dissertation the sampling bottleneck has been empirically shown. Additionally, the JSD score has been evaluated a successful proof of concept to map between workloads. Now the evaluation baselines as introduced in the methodology section can be examined in detail.

The first baseline to meet such that this dissertation can be deemed a success was the default value or out-of-the-box performance value for a given workload. The second, more challenging baseline is the final *argmax* value computed by the original tuning architecture if **all samples** were to be used. To this end, the tuning architecture developed by Barbulescu will serve as the baseline. Finally, the question of whether the sampling bottleneck was able to be reduced needs to be addressed.

When considering figures 17 and 18 it becomes evident that both baselines were met and most importantly, the sampling bottleneck issue has been addressed and solved. In both figures 17 and 18 blue represents the original workload using the OLTP Bench settings A (as introduced in the implementation section) and orange stands for the new workload using the benchmark settings E. In other words, the benchmark settings A (blue) and E (orange) describe the precise parameters used in the experimental set-up for this dissertation.

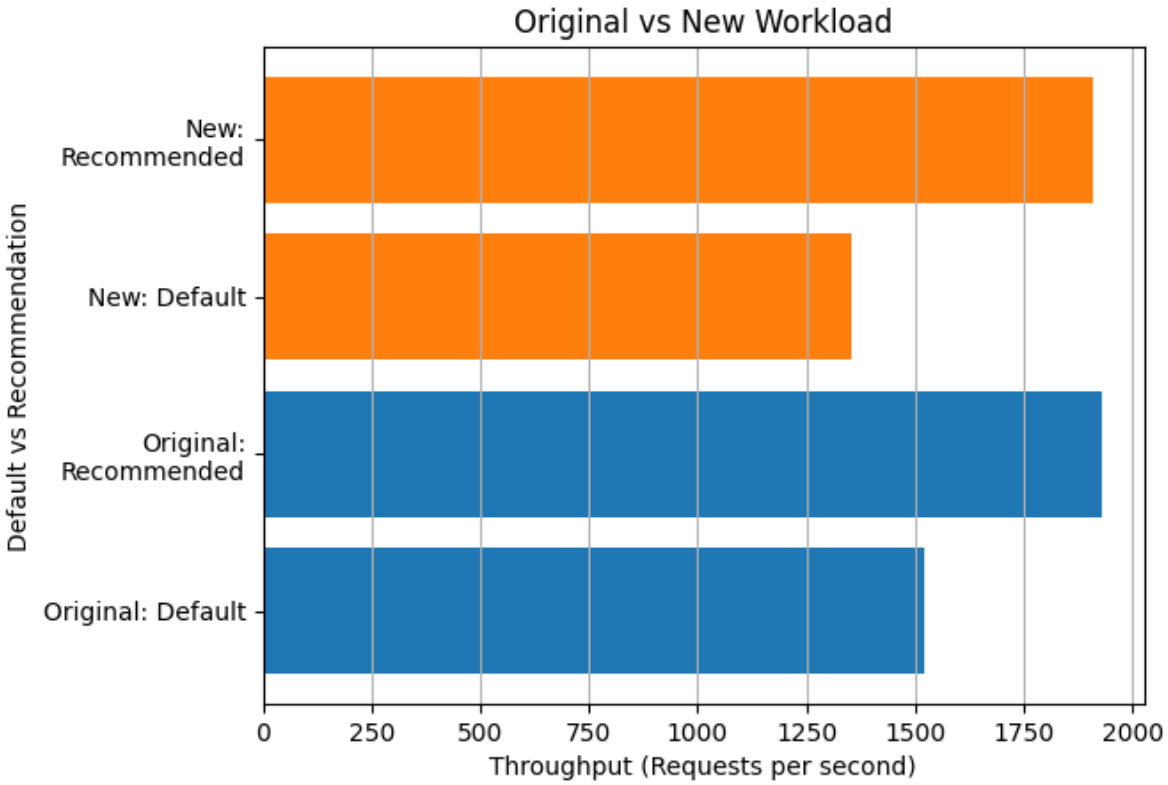


Figure 17: Recommendation for Original vs New Workload

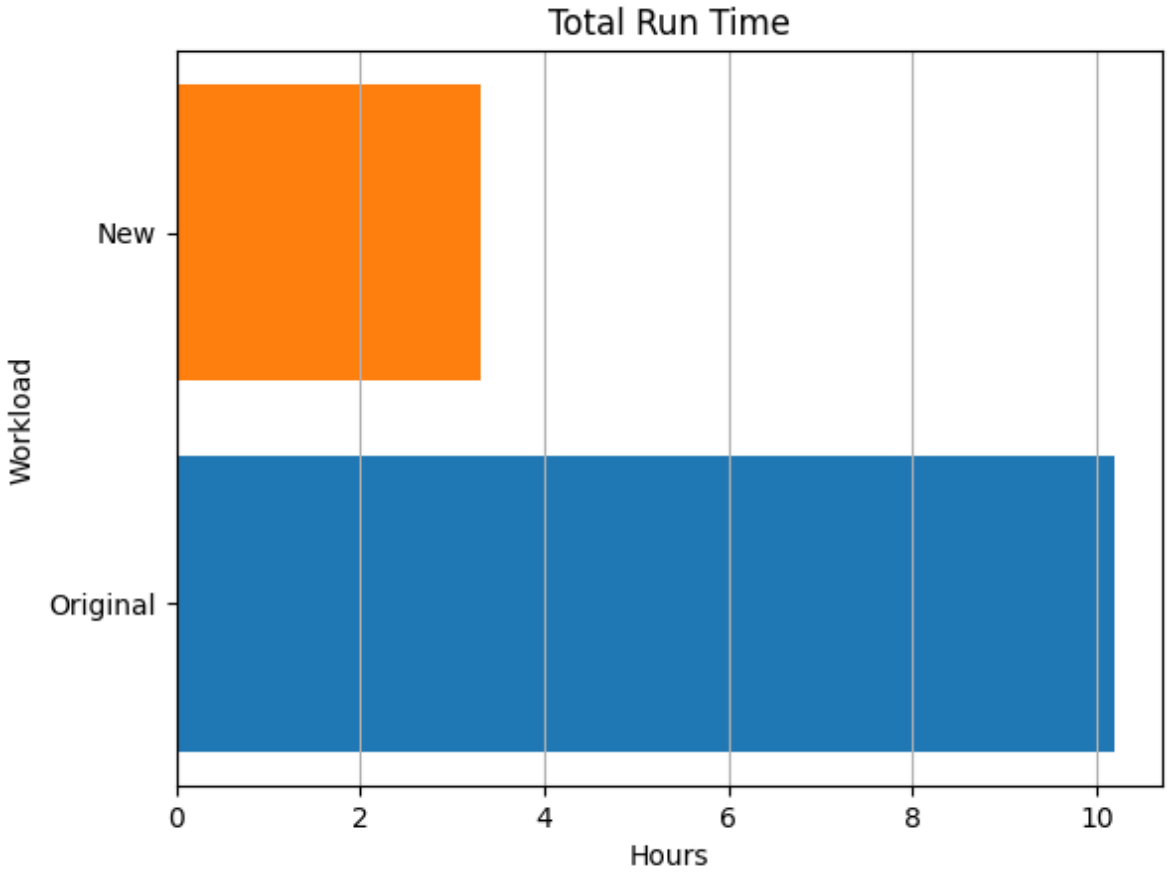


Figure 18: Total Running Time for Original vs New Workload

Based on the weighted JS divergence presented in previous sections the final similarity score $\hat{\theta}$ was calculated to be approximately 0.34 between workloads A and E. As such, the sampling bottleneck was reduced to only $n = 34$ data points for workload E as opposed to $n = 100$ samples for workload A. Consequently, the total run time was cut down to about 3.5 hours for workload E as opposed to the original 10+ hours for workload A. This result can be seen in figure 18 which clearly demonstrates that the novel techniques proposed in this dissertation succeeded in reducing the sampling bottleneck.

When considering figure 17 it becomes evident that the first baseline has been clearly met. The recommendations made by the ensemble model presented in the design section was able to leverage the $n = 34$ data points for workload E (orange) to return competitive database settings that outperform the default settings. More specifically, the default settings allowed for a performance of about 1300 requests per second whereas the settings returned by the ensemble model perform at around 1900 requests per second. This is a definite success for the ensemble model and the sampling bottleneck reduction technique proposed in this dissertation.

However, figure 17 shows that the results are not as evident when it comes to the second baseline. When comparing the $argmax$ performance results of the original workload (blue) against the $argmax$ performance results of the new workload (orange) they seem quite to be quite similar. Both results return around 1900 requests per second. Experimentally, it has been determined that approximately 1940 requests a second is the upper bound based on the experimental set up for this dissertation. Thus, it can be concluded that the recommendations made by the ensemble model are quantitatively as good as the recommendations made by the original BTSTR [4] model when it comes to a single new workload. Said conclusion is based on the observation that workloads A and E have similar upper performance limits when both are tuned using Barbulescu's architecture for a single workload only.

In conclusion, the new sampling function and ensemble model are successful because they significantly reduce the sampling bottleneck while maintaining competitive database settings. As such, a scientifically sound research approach was demonstrated since the baselines which have been defined in the design section were met.

6.4 Ensemble Model

Finally, it is also interesting to take a look at the predictions which the ensemble model made based on workloads A and E. When considering figure 19 it becomes clear that it resembles figure 10 from the design section. This is a desirable output as it shows that the concepts introduced the design section hold when applied with real life data.

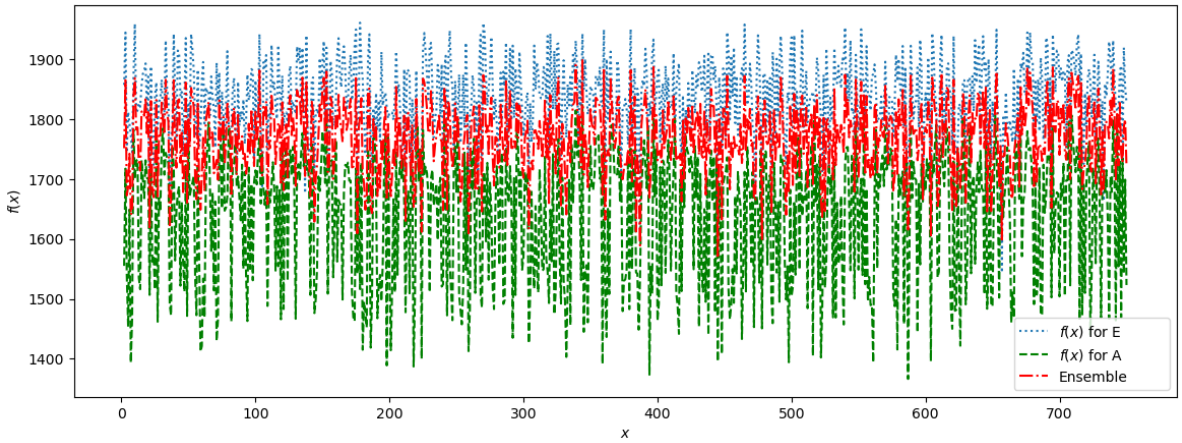


Figure 19: Ensemble Model on Real Data

In figure 19 the red line represents the ensemble model. The blue line represents workload E and and workload A is represented by the green line. Similarly to figure 16 this figure captures the throughput in chronological order. The x-axis represents a single database setting. However, each setting is random sample within the domain of all possible samples. Thus, the regression lines look very choppy. To smooth the regression lines the the database settings (which are a vector of k database setting categories) could be lexicographical sorted or a one dimensional principal component analysis could be applied.

7 Project Management

In the following sections, the project management aspect of this dissertation will be discussed. The main focus will be on agile methodologies and the shift of the research focus at the beginning of term two.

7.1 Deviation from the Specification Report

Since this dissertation is a research-based project the initial specification report planned in term one as the research term and term two as the implementation term. In retrospect, this division made sense and most of the implementation work was done in term two.

However, the main deviation from the specification report is that the research focus changed towards the beginning of term two. The main reason for this change is that the experimental evaluation performed at the beginning of term two concluded that the sampling bottleneck is a bigger issue when it comes to overall tuning time compared to model training time. Thus, the research focus was shifted from transfer learning to the sampling bottleneck.

7.2 Trello

Since a rigid, plan-based approach to this dissertation did not seem feasible anymore after the research focus shifted, a more agile project management approach was needed. To this end, Trello [43] was introduced which is an online Kanban board commonly used for agile software engineering teams.

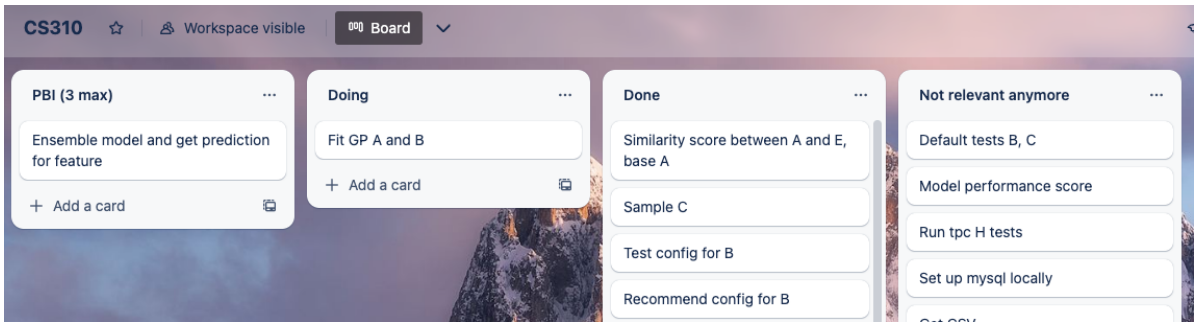


Figure 20: Trello - Online Kanban Software

The key to a successful Kanban board is to not overload the product backlog item (PBI) lane. The PBI lane is also commonly referred to as the to-do list. By regularly

reviewing the items on the PBI lane and moving them to the "not relevant anymore" lane if required, productivity is maximised by only focusing on the most important tasks. This has been a tremendously helpful tool for managing this dissertation as rigid, specification-based thinking had to be replaced quite swiftly.

7.3 Software Development Methodology

Due to the introduction of the Kanban board, the remaining coding aspect of the project has been implemented in an agile way as well. Additionally, progress was discussed in regular meetings with Professor Triantafillou and Meghdad Kurmanji which could be interpreted as "stand-up"-like meetings if viewed from an agile, software engineering perspective.

Additionally, it is worth mentioning that the code developed in this dissertation has been stored on a GitHub repository. As such, the risk of losing code was mitigated in case of laptop theft or other events.

7.4 Legal, Social, Ethical and Professional Issues

As outlined in the original specification report there are no social or ethical issue to consider since no interviews or surveys are conducted. There are also no legal issues to consider as all resources and software products used for this project are open source.

8 Evaluation

In the following sections, the achievements and limitations of this dissertation will be discussed. Then, the lessons learned and potential further work ideas will be presented. This section and dissertation will then end with an overall evaluation of the project as a whole.

8.1 Achievements

The overall goal of this dissertation was to find a novel approach to solving the sampling bottleneck problem as specified in the research statement. To this end, SMART objectives were formulated which were translated into deliverables such as the baselines introduced in the methodology section.

As such, the technical contribution of this dissertation is the research and implementation of the novel sampling function based on the Jensen-Shannon divergence plus the ensemble model built on top of said function. In the end, the research outcome of this dissertation is that the new sampling function, implementation of the JS divergence, and the proposed ensemble model were able to significantly reduce the sampling bottleneck. Moreover, the sampling bottleneck has been reduced while respecting baseline performance levels and thus all research objectives of this dissertation have been successfully achieved.

8.2 Limitations and Future Work

In the following sections, the main limitations of this dissertation will be evaluated. Said limitations will then be used as the basis for suggestions for future research.

8.2.1 Number of Workloads Tested

One of the main limitations of this dissertation is that the number of required workloads to perform Welch's t-test have not been collected in time. As such, the results presented in this dissertation are not as statistically sound as they could be.

However, based on the results shown so far there exists reasonable evidence that the hypothesis presented in this dissertation might hold. Therefore, the first step of any

further research would be to collect more data for different workloads. Then, the test statistics could be calculated and using Welch's t-test it could be empirically determined whether the proposed hypothesis holds or not.

8.2.2 Hardware Dependency

As shown in the implementation section database tuning is highly dependent on the underlying hardware profile. Even though it is difficult to determine what a "standard" hardware profile might look like for a server, it is fair to say that a personal laptop does not qualify as an "enterprise-level" machine.

Thus, any further work based on this dissertation should make sure that the underlying hardware profile qualifies as "enterprise-level". However, to completely disregard the results of this dissertation would be inappropriate. The results presented so far serve as a proof of concept especially if the hypothesis holds and the appropriate hardware profile is selected.

Additionally, it might be an interesting idea as proposed by Kanellis et al. [22] to further experiment with different hardware profiles in the cloud domain. Most cloud platforms offer a variety of hardware profiles to choose from, hence upgrading or downgrading one's current hardware profile can be achieved in a matter of clicks. Cloud computing-based database tuning was outside of the scope of this dissertation in order to reduce complexity.

Hence, an interesting scenario could be to optimise a given workload on one hardware profile. Then, the hardware profile on the cloud platform could be changed. The final goal of this research scenario could be to optimise the same workload again but on a different hardware profile. Here, the JS divergence could be further investigated in this scenario to determine the performance similarity between the original and new hardware profile.

8.2.3 Exploring Other Benchmarks

Another limitation of this dissertation compared to published papers such as OtterTune [46] is that it only uses one benchmark. The majority of published papers use the TPC-C benchmark but most [22, 48, 45] also use the Yahoo! Cloud Serving Benchmark (YCSB)

[6]. Thus, possible further research for this dissertation could be to test the proposed ideas on different benchmarks such as YCSB.

Additionally, it might be interesting if the TPC-C benchmark can be replaced by TPC-E. Even though this is outside the scope of this research area it would undoubtedly be a valuable contribution to the research community if someone developed an open-source, TPC-E inspired testing framework which can be used like OLTP Bench.

8.2.4 Exploring Other Architectures

Lastly, the ML models considered for this dissertation are limited to the Gaussian process regression model and random forest trees. Said models were successful in accomplishing the research objective, however, there exist other ML models that are worth considering.

One of these additional models could be neural networks as proposed by Van Aken et al. in [45]. However, neural networks require excessive amounts of training data so they might not be suited to solve the sampling bottleneck problem. Alternatively, it might interesting to consider a decision tree regression model or a bagging regression model. Both of these can be used in the ensemble model proposed in this dissertation and could be a starting point for further research.

8.3 Lessons Learned

Throughout the course of this dissertation, three main lessons have been learned. Firstly, the scope of any research project should be kept flexible. At the beginning of the project, it is difficult to anticipate results and experiments might nudge the project in a different direction than originally planned.

Secondly, to meet regularly with supervisors and peers to discuss progress. Especially when it feels like the project has been stalling. Verbalizing ideas and hearing other people's insights are invaluable sources of progress.

Lastly, to balance out the time spent writing the report versus time spent experimenting and developing ideas. It might not seem obvious at first but the process of writing the report helps clarify concepts and can lead to new insights about potential avenues to explore.

8.4 Conclusion

In conclusion, the specified research objectives for this dissertation have been met. More precisely, the sampling bottleneck was reduced whilst maintaining the competitiveness of database setting recommendations. This has been achieved by defining a clear research statement at the beginning of the dissertation. Then, a research methodology was introduced based on which results were evaluated. In the main body of the dissertation design and implementation decisions were clearly elucidated. In the end, experimental results were critically evaluated and the achievements and the limitations of this dissertation were discussed.

9 Bibliography

References

[1] From CS352, Accessed: 2022-11-11. URL: https://ianjseath.files.wordpress.com/2009/06/objectives-measures-and-deliverables.pdf.

[2] Pavlo et al. *OLTPBench github repository*. URL: https://github.com/oltpbenchmark/oltpbench (visited on 03/28/2023).

[3] Javed A Aslam and Virgil Pavlu. "Query hardness estimation using Jensen-Shannon divergence among multiple scoring functions". In: *Advances in Information Retrieval: 29th European Conference on IR Research, ECIR 2007, Rome, Italy, April 2-5, 2007. Proceedings 29*. Springer. 2007, pp. 198–209.

[4] George-Octavian Barbulescu and Peter Triantafillou. "Anatomy of Learned Database Tuning with Bayesian Optimization". In: *2022 IEEE 38th International Conference on Data Engineering Workshops (ICDEW)*. IEEE. 2022, pp. 9–15.

[5] Surajit Chaudhuri and Vivek Narasayya. "AutoAdmin "What-If" Index Analysis Utility". In: 27.2 (1998). ISSN: 0163-5808. DOI: 10.1145/276305.276337.

[6] Brian F. Cooper et al. "Benchmarking Cloud Serving Systems with YCSB". In: 2010. ISBN: 9781450300360. DOI: 10.1145/1807128.1807152. URL: https://doi.org/10.1145/1807128.1807152.

[7] Aparna Dhinakaran. *Using Statistical Distances for Machine Learning Observability*. URL: https://towardsdatascience.com/using-statistical-distance-metrics-for-machine-learning-observability-4c874cded78 (visited on 03/23/2023).

[8] Djellel Eddine Difallah et al. "Oltp-bench: An extensible testbed for benchmarking relational databases". In: *Proceedings of the VLDB Endowment* 7.4 (2013), pp. 277–288.

[9] R Elmasri et al. *Fundamentals of Database Systems*. 7th ed. Springer, 2015. ISBN: 978-0133970777.

[10] Stats Exchange. *Understanding the stars and bars formula*. URL: https://math.stackexchange.com/questions/3585245/understanding-the-stars-and-bars-formula (visited on 03/25/2023).

[11] Stats Exchange. *Why is Euclidean distance not a good metric in high dimensions?* URL: https://stats.stackexchange.com/questions/99171/why-

is-euclidean-distance-not-a-good-metric-in-high-dimensions (visited on 03/28/2023).

[12] Google. *Google Scholar*. URL: https://scholar.google.com/ (visited on 03/25/2023).

[13] M. Grootendorst. *9 Distance Measures in Data Science*. URL: https://towardsdatascience.com/9-distance-measures-in-data-science-918109d069fa (visited on 03/23/2023).

[14] M. Hammer. "Self-Adaptive Automatic Data Base Design". In: *Proceedings of the June 13-16, 1977, National Computer Conference*. AFIPS '77. Dallas, Texas: Association for Computing Machinery, 1977, pp. 123–129. ISBN: 9781450379144. DOI: 10.1145/1499402.1499430. URL: https://doi.org/10.1145/1499402.1499430.

[15] M. Hammer and A. Chan. "Index Selection in a Self-Adaptive Data Base Management System". In: *Proceedings of the 1976 ACM SIGMOD International Conference on Management of Data*. Association for Computing Machinery, 1976. ISBN: 9781450347297. DOI: 10.1145/509383.509385. URL: https://doi.org/10.1145/509383.509385.

[16] M. Hammer and B. Niamir. "A Heuristic Approach to Attribute Partitioning". In: *Proceedings of the 1979 ACM SIGMOD International Conference on Management of Data*. New York, NY, USA: Association for Computing Machinery, 1979. ISBN: 089791001X. DOI: 10.1145/582095.582110. URL: https://doi.org/10.1145/582095.582110.

[17] D Howell. "Fundamental Statistics For the Behavioral Sciences". In: (2008).

[18] OnGres Inc. *POSTGRESQLCO.NF Parameter Documentation*. URL: https://postgresqlco.nf/doc/en/param/ (visited on 03/23/2023).

[19] M. Y. L. Ip, L. V. Saxton, and V. V. Raghavan. "On the Selection of an Optimal Set of Indexes". In: *IEEE Trans. Softw. Eng.* 9.2 (1983), pp. 135–143. ISSN: 0098-5589. DOI: 10.1109/TSE.1983.236458. URL: https://doi.org/10.1109/TSE.1983.236458.

[20] Stephen Joe and Frances Y Kuo. "Notes on generating Sobol sequences". In: *ACM Transactions on Mathematical Software (TOMS)* 29.1 (2008), pp. 49–57.

[21] Jupyter. *Jupyter*. URL: https://jupyter.org/ (visited on 03/25/2023).

[22] Konstantinos Kanellis, Ramnatthan Alagappan, and Shivaram Venkataraman. "Too many knobs to tune? towards faster database tuning by pre-selecting im-

portant knobs". In: *12th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 20)*. 2020.

[23] Vijay Kotu and Bala Deshpande. "Chapter 2 - Data Mining Process". In: *Predictive Analytics and Data Mining*. Ed. by Vijay Kotu and Bala Deshpande. Boston: Morgan Kaufmann, 2015, pp. 17–36. ISBN: 978-0-12-801460-8. DOI: https://doi.org/10.1016/B978-0-12-801460-8.00002-1. URL: https://www.sciencedirect.com/science/article/pii/B9780128014608000021.

[24] Meghdad Kurmanji and Peter Triantafillou. "Detect, Distill and Update: Learned DB Systems Facing Out of Distribution Data". In: *arXiv preprint arXiv:2210.05508* (2022).

[25] Kim Jong Hae Kwak Sang Gyu. "Central limit theorem: the cornerstone of modern statistics". In: *kja* 70.2 (2017), pp. 144–156. DOI: 10.4097/kjae.2017.70.2.144. eprint: http://www.e-sciencecentral.org/articles/?scid=1156667. URL: http://www.e-sciencecentral.org/articles/?scid=1156667.

[26] Eva Kwan et al. "Automatic configuration for IBM DB2 universal database". In: *Proc. of IBM Perf Technical Report* (2002).

[27] Michael D. McKay, Richard J. Beckman, and William J. Conover. "A Comparison of Three Methods for Selecting Values of Input Variables in the Analysis of Output From a Computer Code". In: *Technometrics* 42 (2000), pp. 55–61.

[28] M.L. Menéndez et al. "The Jensen-Shannon divergence". In: *Journal of the Franklin Institute* 334.2 (1997), pp. 307–318. ISSN: 0016-0032. DOI: https://doi.org/10.1016/S0016-0032(96)00063-4. URL: https://www.sciencedirect.com/science/article/pii/S0016003296000634.

[29] Ravi Mukkamala, Steven C. Bruell, and Roger K. Shultz. "Design of partially replicated distributed database systems: an integrated methodology". In: *Measurement and Modeling of Computer Systems*. 1988.

[30] Harald Niederreiter. *Random Number Generation and Quasi-Monte Carlo Methods*. USA: Society for Industrial and Applied Mathematics, 1992. ISBN: 0898712955.

[31] Frank Nielsen. "On the Jensen–Shannon Symmetrization of Distances Relying on Abstract Means". In: *Entropy* 21.5 (2019). ISSN: 1099-4300. URL: https://www.mdpi.com/1099-4300/21/5/485.

[32] Numpy. *Numpy*. URL: https://numpy.org/ (visited on 03/25/2023).

[33] Oracle. *How MySQL Uses Memory*. URL: https://dev.mysql.com/doc/refman/8.0/en/memory-use.html (visited on 03/25/2023).

[34] Pandas. *Pandas*. URL: https://pandas.pydata.org/ (visited on 03/25/2023).

[35] Andrew Pavlo et al. "External vs. internal: an essay on machine learning agents for autonomous database management systems". In: *IEEE bulletin* 42.2 (2019).

[36] Andrew Pavlo et al. "Self-Driving Database Management Systems." In: *CIDR*. Vol. 4. 2017, p. 1.

[37] PostgreSQL. *Tuning Your PostgreSQL Server*. URL: https://wiki.postgresql.org/wiki/Tuning_Your_PostgreSQL_Server (visited on 03/23/2023).

[38] Scipy. *Scipy*. URL: https://scipy.org/ (visited on 03/25/2023).

[39] Sklearn. *Sklearn*. URL: https://scikit-learn.org/stable/ (visited on 03/25/2023).

[40] TPC. *HammerDB github repository*. URL: https://github.com/TPC-Council/HammerDB (visited on 03/28/2023).

[41] TPC. *TPC-C*. URL: https://www.tpc.org/tpcc/ (visited on 03/28/2023).

[42] TPC. *TPC-E*. URL: https://www.tpc.org/tpce/ (visited on 03/28/2023).

[43] Trello. *Trello: Manage Your Team's Projects From Anywhere*. URL: https://trello.com (visited on 03/25/2023).

[44] Pennsylvania State University. *Tests of the Equality of Two Means*. URL: https://online.stat.psu.edu/stat415/lesson/11 (visited on 03/23/2023).

[45] Dana Van Aken et al. "An inquiry into machine learning-based automatic configuration tuning services on real-world database management systems". In: *Proceedings of the VLDB Endowment* 14.7 (2021), pp. 1241–1253.

[46] Dana Van Aken et al. "Automatic database management system tuning through large-scale machine learning". In: *Proceedings of the 2017 ACM international conference on management of data*. 2017, pp. 1009–1024.

[47] Eric W Weisstein. "Np-hard problem". In: *From MathWorld–A Wolfram Web Resource. http://mathworld. wolfram. com/NP-HardProblem. html* (2009).

[48] Ji Zhang et al. "An end-to-end automatic cloud database tuning system using deep reinforcement learning". In: *Proceedings of the 2019 International Conference on Management of Data*. 2019, pp. 415–432.

[49] Xinyi Zhang et al. "Restune: Resource oriented tuning boosted by meta-learning for cloud databases". In: *Proceedings of the 2021 International Conference on Management of Data*. 2021, pp. 2102–2114.

[50] Zhe Zhang. *An Automatic Source Code Generation Tool for OLTP Database Benchmarks*. 2018.

[51] Conghuan Zheng, Zuohua Ding, and Jueliang Hu. "Self-tuning performance of database systems with neural network". In: *International Conference on Intelligent Computing*. Springer. 2014, pp. 1–12.

[52] Yuqing Zhu et al. "Bestconfig: tapping the performance potential of systems via automatic configuration tuning". In: *Proceedings of the 2017 Symposium on Cloud Computing*. 2017, pp. 338–350.

A Specification Report

CS310 Project Specification:
Transfer learning for OLTP database tuning

Valentin Kodderitzsch (u1931737)

October 2022

Contents

1 Problem Statement

The aim of this project is to add to the current transfer learning (TL) research landscape in the domain of database management systems (DBMS) tuning via machine learning (ML) models [3]. We are concerned with the distillation and knowledge transfer from one optimized ML model to a second ML model [2].

Both models try to find the most competitive DBMS settings for a predefined, static DBMS workload. A workload is just a collection of CRUD[1] statements run against a database [4]. In our case we are concerned about Online Transaction Processing (OLTP) database tuning. The crux of our problem is that the two workloads are different. In short, the goal is to recommend the most competitive DBMS settings for the second workload by leveraging the tuning knowledge encapsulated in the first, already optimized model to avoid training the second model from scratch.

2 Research

Before exploring why the computer science community is interested in DBMS tuning it worth wile to clarify what exactly DBMS tuning is. At its core DBMS tuning is about achieving the best performance metric after some form of a stress test is executed against the database for different DBMS settings. The most common performance metrics are the DBMS throughput or latency. The type of stress test depends on the type of database. However, the most common test frameworks are TCP-C, YCSB and TCP-H [4]. When it comes to DBMS settings they are usually divided into developer level and admin level settings. Developer level settings are mostly concerned with query plans, i.e. the exact ordering of joins for instance. Admin level settings include physical design changes such as manipulating indexes, hardware resource changes and database knob setting changes. Commonly manipulated knob settings are the log file size or the buffer pool size as well as caching policies. All admin level settings affect the overall performance of the DBMS. The problem of tuning knob settings is especially interesting as it is an $\mathcal{NP}$-hard problem [4]. As such leveraging ML models to approximate the problem becomes a promising proposition for the computer science community to explore.

There are 2 main approaches to DBMS tuning as outlined by the authors of [3]. Namely, internal vs external DBMS tuning agents. An agent is usually a bigger software component that can implement changes on its own. Compared to an internal agent the external agent views the DBMS as a "black-box". This means that the agent only receives data about the DBMS via a third party application such as JDBC[2]. The model has no "knowledge" about the inner workings of the DBMS. The internal agents on the other hand is part of the DBMS architecture and can has access to more information. As such internal agents are more "knowledgeable". However, they are also more difficult to implement. Examples of external agents are BTSTR [1], OtterTune [4] and ResTune [5]. The authors of [3] also mentioned NoisePage as an example for internal tuning.

Within the context of DBMS tuning there is a research area which is interested in transfer learning. TL as described by the authors of [2], albeit in a different

[1] Create, read, update, delete

[2] A Java API to communicate with databases.

context, is the process of distilling and transferring the knowledge from a first model to a second model. This second model aims to solve a similar task compared to the first model. One of the main goals of TL is to learn the second task faster than it took to learn the first task. Ideally, the second model should also perform better at the second task compared to the first model and the first task. An analogy described by the authors of [6] is learning how to ride a motorcycle after you have learned how to ride a bicycle. Additionally, there are 2 more variations to TL. The first one is where you "unlearn" the first task, i.e. you learned how to ride a motorcycle very quickly, however, you have now forgotten how to ride a bicycle. The second variation is that the model is now able to solve both task, i.e. you can ride a bicycle and a motorcycle in the end. Both variations have different constraints and implementation challenges.

One paper which is concerned with TL in the context of DBMS tuning has been co-authored by Meghdad Kurmanji[3]. It is titled DDUp[4] but has not been published yet. The main concern of the DDUp paper is to transfer learning from one model to another when the distribution of our data set changes. In the context of a DBMS this means that new records have been inserted but we are still concerned about performance metrics such as approximate query processing (AQP) time. One of the main differences between our problem statement and DDUp is that we are concerned with workload changes instead of just data distribution changes, i.e. all CRUD statements compared to insert statements only. As such we will be working on OLTP databases whereas DDUp focused on OLAP databases. Additionally, we are specifically interested in finding the most competitive database knob settings. We are also not concerned with maintaining the original model whereas DDUp tries to avoid "unlearning".

In conclusion, the exact gap we are trying to fill in the current research landscape is how to transfer learning from one workload optimization ML model to another model when the workload changes. Even though papers such as OtterTune [4] or BTSTR [1] have touched upon TL or "workload mapping" their efforts have been focused on creating an end-to-end automated DBMS tuning agents. Our efforts on the other hand are solely focused on TL in the context of OLTP database tuning and will as such provide novel insights for the computer science community.

3 Thesis Objective

As opposed to the projects from authors of BTSTR [1] and OtterTune [4] we do not aim to create an updated end-to-end automated DBMS tuning system. Instead the goal of this thesis is to propose a novel TL approach in the domain of OLTP workload optimization. As such we want to focus our development efforts on the workload optimization and the TL modules instead of an entire

[3]PhD Candidate at the University of Warwick

[4]Detect, Distill and Update: Learned DB Systems Facing Out of Distribution Data

DBMS tuning pipeline. Those 2 modules will then be thoroughly tested with special emphasis on the TL one.

3.1 Workload optimization

To achieve the workload optimization milestone the following steps need to be completed.

1. Set up the DBMS environment
2. Find a workload test framework
3. Identify the best ML model which also allows for TL
4. Choose a dimensionality reduction technique for the DB settings
5. Create a ML model to find optimal DB setting for a specific workload A

For this milestone we can make use of the research presented by the authors of OtterTune [4], BTSTR [1] and possibly Restune [5].

3.2 Transfer Learning

For the transfer learning milestone to be achieved the following steps need to be completed.

1. Identify the best TL approach
2. Create a second ML model with distilled knowledge from the first model
3. Optimize the second model for a new workload B

Here we can make use of the ideas presented in the 2 papers co-authored by Meghdad Kurmanji[5][6] which are yet to be published. Papers [2, 6] provide a general overview of the TL landscape. Additionally, it is worth mentioning that we don't want our TL module to maintain the original model as described in Meghdad Kurmanji's paper. We only care about providing the most competitive DB settings for the new workload B.

3.3 Testing

To test our work we can follow the steps outlined in the BTSTR [1] paper. This means running a target workload against a set database server to check the workloads target metrics (e.g. throughput, latency). Additionally, it would make sense to run the following 2 TL specific tests.

1. Compare the tuning time for workload B with TL vs the tuning time of workload B without TL

[5]Detect, Distill and Update: Learned DB Systems Facing Out of Distribution Data
[6]The brainy student: Scalable deep unlearning by selectively disobeying the teacher

2. Compare the tuning performance of the model with TL vs the tuning performance of the model without TL

3.4 Possible extensions

A possible extension of the project could be to adopt out TL insights to evolving DB deployments such as changing hardware profiles. However, this step will only be reached once the previously mentioned steps have been successfully completed.

4 Methods

As mentioned in the Thesis Objective section the main approach to implementing this project is to adopt and further develop parts of our project from previous work in the area. More specifically, we will base the workload optimization module on the work presented in BTSTR [1]. Our TL module will be based on DDUp. Tesing will be based on both papers. Additional papers will be considered as needed. The authors of BTSTR and DDUp are both Warwick students which means that I will be able to reach out to them to ask for guidance. Details on how the project will be guided to completion can be found in the project management section.

4.1 Resources

In terms of resources we can again make use of the approaches presented by the authors of BTSTR and DDUp. The exact list will most likely change throughout the course of this project. However, the following list should give a good overview.

1. Access to a database server running a DBMS such as PostgreSQL [1]
2. Access to a DBMS test framework such as TCP-C [1]
3. A selection of ML libraries such as scikit-learn or TensorFlow

For item 1) and 2) I should be able to consult with supervisor and the author of [1] to figure out the details. For item 3) I will reach out to the authors of DDup and BTSTR to find out the exact libraries and model choices I should make for this project.

5 Risk Assessment

The main risk of this project is either running out of time or realizing that the original approach was not valid. As such it is important to have regular meetings to discuss the pace and trajectory of this project. A provisional meeting schedule will be laid out in the project management section.

The second biggest risk might be difficulties accessing resources 1) and 2). However, as the author of BTSTR had to access the same resources I am confident that it won't be an issue for this project either.

6 Project Management

The following section outlines how I intend on carrying out this project. The last but most important deadline is May 2nd 2023 which is when the final report is due. Before that there are 3 more deadlines to help with the progress of the project. The first deadline is due on October 13th 2022 which is this specification report. The next deadline is the progress report which is due about 6.5 weeks later on November 28th 2022. The last pre final report deadline is due between March 6th and 17th, depending on your assigned presentation date. This means that by March I need to have produced tangible results in terms of functioning code for my assessors and peers. After the presentation I will have about 2 months to convert my notes, submissions and code into a final report. This project will take about 7 months from initiation to completion.

The 4 key submissions dates are captured in the following table.

Item	**Date**	**Goal**
Specification Report	Oct 13 2022 (T1W2)	Scope problem statement
Progress Report	Nov 28 2022 (T1W9)	Start coding
Presentation	Mar 6 to 17 2023 (T2W9-10)	Complete coding
Final Report	May 2 2023 (T3W2)	Write up

6.1 Timeline

The following table aims to provide more detail about the overall project timeline. It will be reviewed regularly throughout the course of the project.

Term	Week	Goal
1	1	Project initiation
1	2	Project specification submission
1	3	Explore existing literature
1	4	
1	5	Learn about DBMS tuning implementation details
1	6	
1	7	Learn about TL implementation details
1	8	Prepare progress report
1	9	Progress report submission
1	10	Learn about TL implementation details
Winter break		Final report write up based on term 1
2	1	Implement workload optimization module
2	2	
2	3	
2	4	Implement transfer learning module
2	5	
2	6	
2	7	
2	8	Testing
2	9	Testing and presentation preperation
2	10	Project presentation
Spring break		Final report write up
3	1	Final report write up
3	2	Final report submission

Figure 1: Project timeline overview

6.2 Guidance

As mentioned in previous sections the success of this project depends on keeping the right trajectory and pace over the course of the next 7 months. Therefore,

it is essential that I am in regular contact with my supervisor and the authors of the DDUp and BTSTR papers.

I intend on speaking with my supervisor at least once a month. This will allow me to present bigger chunks of work I have completed. As such our discussions will be focused more on the general trajectory and research areas I should focus on.

I am hoping to speak to the authors of DDUp or BTSTR every 2 weeks. This will of course depend on their schedule. My goal is to discuss ideas presented in their papers with them. Additionally, I am looking for concrete guidance in terms of implementation details. For instance which ML models or libraries to choose. I am also hoping to learn more about my project through hearing about their experiences. Finally, I am hoping that having a meeting every 2 weeks will help me maintain a reasonable pace.

6.3 Documentation

This project will be documented using Google Docs and Google Drive. Throughout the project I will keep a weekly log of all the task I achieved during the week plus comments. This will hopefully make the progress and final report easier to write. Additionally, I will record any meeting notes on Google Docs. I will also continue maintaining my reading list. This list is stored on my Google Drive and is separated into "read" and "to be read" folders. All papers in the "read" folder contain a summary such that they can easily be referred back to in the future.

6.4 Version Control

This project will use Git for version control and the remote repositories will be stored on GitHub.

7 Legal, social, ethical and professional issues

There are no social and ethical[7] issues to consider with this project as no work with other people (e.g. interviews) is required.

As the project aims to add to the current TL landscaping using open-source software resources there are also no legal or professional issues to consider.

[7] https://warwick.ac.uk/fac/sci/dcs/teaching/material/cs310/ethics

8 Bibliography

References

[1] George-Octavian Barbulescu and Peter Triantafillou. "Anatomy of Learned Database Tuning with Bayesian Optimization". In: *2022 IEEE 38th International Conference on Data Engineering Workshops (ICDEW)*. IEEE. 2022, pp. 9–15.

[2] Geoffrey Hinton, Oriol Vinyals, Jeff Dean, et al. "Distilling the knowledge in a neural network". In: *arXiv preprint arXiv:1503.02531* 2.7 (2015).

[3] Andrew Pavlo et al. "External vs. internal: an essay on machine learning agents for autonomous database management systems". In: *IEEE bulletin* 42.2 (2019).

[4] Dana Van Aken et al. "Automatic database management system tuning through large-scale machine learning". In: *Proceedings of the 2017 ACM international conference on management of data*. 2017, pp. 1009–1024.

[5] Xinyi Zhang et al. "Restune: Resource oriented tuning boosted by meta-learning for cloud databases". In: *Proceedings of the 2021 International Conference on Management of Data*. 2021, pp. 2102–2114.

[6] Fuzhen Zhuang et al. "A comprehensive survey on transfer learning". In: *Proceedings of the IEEE* 109.1 (2020), pp. 43–76.

B Progress Report

CS310 Progress Report:
Transfer learning for OLTP database tuning

Valentin Kodderitzsch (u1931737)

November 2022

Contents

1 Introduction

As stated in the original specification report the goal of this research project is to find a novel transfer learning approach in the context of DBMS[1] tuning. DBMS tuning as described the authors of [13] is the process of systematically choosing the best DBMS settings to "improve a DBMS's operations based on some objective function (e.g., faster execution, lower costs, better availability)". Such settings are usually divided into developer level versus admin level settings such as the log file size or the buffer pool size. Said settings are also often called database knob settings. In terms of data processing systems there exist OLTP[2] and OLAP[3] database systems. To narrow down the scope of this project we chose to focus on tuning admin level settings of OLTP systems.

Historically speaking there have been multiple approaches to DBMS tuning but the most recent research efforts are focused around leveraging different machine learning techniques including neural networks [18]. As such we can explore topics that are not constrained to but popular within the neural network research community such as transfer learning [19]. Traditional machine learning assumes that training and testing data have the same underlying distribution [10]. If the distribution was to change, however, the model needs to be trained again. This creates an incentive to leverage knowledge from previous training sessions for the current one as training a model is resource expensive.

2 Problem Statement

Finding a novel transfer learning approach is a challenging endeavour insofar as it requires you to set up an entire DBMS tuning pipeline before you can experiment with different transfer learning techniques. However, it is an achievable challenge as there exist a number of open-source DBMS pipelines such as OtterTune [15]. Therefore, the value proposition of this project is the development of a better performing transfer learning mechanism compared to the existing solutions.

2.1 Possible SMART Objectives

SMART objectives as specified by [1] and presented in CS352[4] are specific, measurable, achievable and relevant objectives. They represent achievements which can be tracked against a graph and do not specify any implementation details. However, concrete plans or milestones, i.e. deliverables can be derived from well conceived SMART objectives. Possible SMART objectives for this project could be

[1] Database management systems
[2] Online transaction processing
[3] Online analytical processing
[4] https://warwick.ac.uk/fac/sci/dcs/teaching/material/cs352/

1. To decrease the convergence time of the new transfer learning module by at least 10% compared to the original workload mapping module, by March 3rd 2023 (or at least 3 days before the term 2 presentation)
2. To decrease the latency for the new workload by at least 10% compared to the original solution, by March 3rd 2023 (or at least 3 days before the term 2 presentation)
3. To increase the throughput for the new workload by at least 10% compared to the original solution, by March 3rd 2023 (or at least 3 days before the term 2 presentation)

Compared to the problem statement presented in the original specification report we now have much more tangible transfer learning objectives. However, it has become apparent in the last 7 weeks that deliverables will change with time. As such it is important to maintain a flexible project management approach regarding deliverables as long as the overall transfer learning objectives are clear. This will be further discussed in the "Updated Project Plan" section.

3 Project State

The following section will describe the project state compared against the milestones, i.e. deliverables set out in the original specification report regarding the workload optimization module. However, as described in the "Unforeseen Challenges" chapter the original milestones have changed a bit. Nonetheless, it can be assumed that the deliverables for the transfer learning and the testing remain unchanged for now.

3.1 Milestone 1

The first milestone as originally discussed was to set up a DBMS environment on a local machine. To do so an appropriate DBMS vendor needed to be identified. In the current DBMS research literature PostgreSQL [2] and MySQL [3] are the 2 most prominent DBMS vendors. Over the past 2 decades both of them have been able to establishes themselves as reliable open-source vendors with abundant community support.

For this project MySQL was chosen as the general consensus is that they are both very similar but MySQL is easier to set up and get started with [4]. MySQL was set up using Homebrew [5] and by following the official documentation [6].

```
# Install using homebrew
$ brew install mysql

# Start the MySQL service
$ brew services start mysql

# Set root MySQL password
```

```
$ mysqladmin -u root password 'secretpassword'

# Access MySQL
$ mysql -u root -p
```

To access the database more easily a graphical user interface client such as SequelPro[5] can be used.

3.2 Milestone 2

The second milestone was to find an appropriate DBMS test framework. As explained in the original specification report a DBMS is tuned against a static workload which is just a collection of CRUD[6] statements executed against the database.

Since 1992 the Transaction Processing Performance Council has been providing the TPC-C benchmark which has become an industry and research standard. However, the author of [17] argues that TPC-C is outdated and ought to be replaced with TPC-E. Unfortunately, the problem with official TPC benchmarks is that they are extremely difficult to run due to the lack of official documentation and community support.

Therefore, OLTPBench [11] which is an open-source software developed by the authors of OtterTune [15] is a promising solution. Even though OLTPBench does not offer a TPC-E benchmark it still provides a TPC-C benchmark. An alternative to OLTBench is the percona lab benchmark [7] as referenced by the authors of CDBTune [16]. However, OLTPBench still seems like a better alternative as it has better documentation and provides numerous different benchmarks compared to the percona lab benchmark which only offers TPC-C.

3.3 Milestone 3

The third milestone for this term was to identify the best machine learning model for transfer learning. However, there was a slight complication with this milestone which will be discussed in the next chapter.

3.4 Milestone 4

The fourth milestone was to choose a dimensionality reduction technique for the plethora of knob settings presented to the machine learning algorithm. Existing DBMS tuning pipelines such as OtterTune leverage factor analysis and L1 regression for feature selection. However, a better solution might be to compute the R^2 score of each knob setting in combination with a Latin Hypercube Sampling technique for the data generation process as outlined by the authors of [12].

[5] https://www.sequelpro.com/
[6] Create, read, update, delete

3.5 Milestone 5

The last milestone was to train a machine learning model for a specific workload. However, as this milestone depends on milestone 3 it could not be completed as originally planned.

3.6 Evaluation

Compared to the original specifications good progress has been made as 3 out of 5 milestones have been reached. Additionally, numerous papers in the surrounding literature have been investigated. Especially papers regarding transfer learning such as [10, 19]. However, there have been some unforeseen challenges which will be discussed in the next chapter.

4 Unforeseen Challenges

The main challenge so far was that the project deliverables got updated in the middle of term 1. To experiment with transfer learning you need a working DBMS tuning pipeline. This can be achieved by either grabbing an existing, open-source solution from the literature or by building one from scratch. Due to the lack of communication from my part I assumed that the DBMS pipeline for this project needed to be build from scratch. However, it became apparent during the second supervisor meeting that building on top of an existing DBMS solution might be a better approach. As such the third workload optimization milestone, i.e. identifying a fitting machine learning module which allows for transfer learning has been replaced with "finding an existing DBMS tuning pipeline which uses neural networks".

The main advantage of building on top of an existing solution is that the data generation part via the workload test framework is already taken care of. Fortunately, the authors of OtterTune published a new paper [14] last year in which they describe an updated DBMS tuning pipeline which leverages neural networks as well the original Gaussian process regression model. This means that the updated milestone has been achieved. As the code for OtterTune is publicly available on GitHub [8] we can now investigate its suitability for this project. This is especially important as the code has been archived in 2022 and the last edit has been made in 2020. Therefore a number of key questions must be answered such as

1. Does the code still run at all
2. If it runs is it outdated or usable
3. If it is usable can their data generation module be used for this project
4. If it is usable can their neural network be used for this project

5 Updated Project Plan

As correctly identified in the original specification report one of the main risks of this project is moving into the wrong direction for too long. Unfortunately, this has happened as outlined in the last chapter. The consequences have been manageable, however, it became apparent that the project plan needs updating.

5.1 Guidance

The guidance section of the specification report mentions supervisor meetings at least once a month and implementation level meetings every 2 weeks. However, these time frames seem a bit too long given that there are now about 5 months left for this project. Additionally, it became obvious that specification changes are inevitable. Therefore, a more agile project planning approach might be more suitable. In terms of actionable steps this means

1. Planning each week as a sprint with clearly defined deliverables to be achieved
2. Having very brief stand ups (via email, teams or face-to-face) at the end of each week
3. Deciding during stand ups if a longer meeting is required

5.2 Methods and Resources

The original specifications states that the workload optimization module will be based on the findings presented in the BTSTR paper [9]. Even though the OtterTune solution requires an investigation it will most likely serve as the new foundation of this project. Consequently, the provisional list of resources presented in the specification report can be substituted by the OtterTune code base which covers the DBMS, a data generation tool, the workload optimization and the transfer learning modules.

Figure 1 gives an overview of the DBMS tuning pipeline architecture as presented in the new OtterTune paper [18]. As previously described the plan is to thoroughly investigate how the controller and the tuning manager interact with the target DBMS. Once the inner workings have been understood we can focus our efforts on the tuning manager to build on top of step 4. As such figure 1 can be seen as preliminary design for this project.

5.3 Updated Timetable

Due to the deliverables changes the original timetable needs to be updated. It can be found in figure 2. The main difference compared to the original timetable is that from week 10 onward the deliverables are focused on OtterTune. Additionally, there have been 3 more items added to the winter break period to ensure that the project finishes on time. It also important to note that the

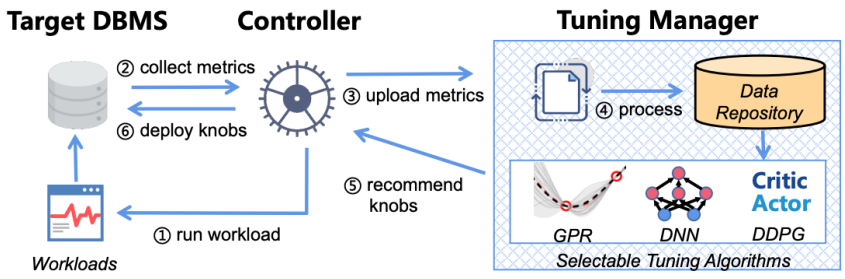


Figure 1: The new OtterTune architecture overview from [18]

deliverables for the first 7 weeks of term 2 are proposed deliverables and not set in stone. This is a consequence of our term 1 learnings that deliverables might change and that continuous progress discussion is vital for the success of this project.

Term	Week	Goal
1	1	Project initiation
1	2	Project specification submission
1	3	Explore existing literature
1	4	
1	5	Learn about DBMS tuning implementation details
1	6	
1	7	Learn about TL implementation details
1	8	Prepare progress report
1	9	Progress report submission
1	10	Investigate OtterTune
Winter break		1) Discuss OtterTune investigation 2) Plan next set of deliverables 3) Continue reading about TL 4) Final report write up based on term 1
2	1	Isolate the NN model from OtterTune
2	2	
2	3	
2	4	Implement new transfer learning technique
2	5	
2	6	
2	7	
2	8	Testing
2	9	Testing and presentation preperation
2	10	Project presentation
Spring break		Final report write up
3	1	Final report write up
3	2	Final report submission

Figure 2: Updated timeline

6 Bibliography

References

[1] From CS352, Accessed: 2022-11-11. URL: https://ianjseath.files.wordpress.com/2009/06/objectives-measures-and-deliverables.pdf.

[2] Accessed: 2022-11-1. URL: https://www.postgresql.org/.

[3] Accessed: 2022-11-1. URL: https://dev.mysql.com/.

[4] Accessed: 2022-11-1. URL: https://www.integrate.io/blog/postgresql-vs-mysql-which-one-is-better-for-your-use-case.

[5] Accessed: 2022-11-1. URL: https://formulae.brew.sh/formula/mysql.

[6] Accessed: 2022-11-1. URL: https://dev.mysql.com/doc/mysql-installation-excerpt/8.0/en/macos-installation-notes.html.

[7] Accessed: 2022-11-1. URL: https://github.com/Percona-Lab/tpcc-mysql.

[8] Accessed: 2022-11-23. URL: https://github.com/cmu-db/ottertune.

[9] George-Octavian Barbulescu and Peter Triantafillou. "Anatomy of Learned Database Tuning with Bayesian Optimization". In: *2022 IEEE 38th International Conference on Data Engineering Workshops (ICDEW)*. IEEE. 2022, pp. 9–15.

[10] Wenyuan Dai et al. "Eigentransfer: a unified framework for transfer learning". In: *Proceedings of the 26th Annual International Conference on Machine Learning*. 2009, pp. 193–200.

[11] Djellel Eddine Difallah et al. "Oltp-bench: An extensible testbed for benchmarking relational databases". In: *Proceedings of the VLDB Endowment* 7.4 (2013), pp. 277–288.

[12] Konstantinos Kanellis, Ramnatthan Alagappan, and Shivaram Venkataraman. "Too many knobs to tune? towards faster database tuning by preselecting important knobs". In: *12th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 20)*. 2020.

[13] Andrew Pavlo et al. "External vs. internal: an essay on machine learning agents for autonomous database management systems". In: *IEEE bulletin* 42.2 (2019).

[14] Dana Van Aken et al. "An inquiry into machine learning-based automatic configuration tuning services on real-world database management systems". In: *Proceedings of the VLDB Endowment* 14.7 (2021), pp. 1241–1253.

[15] Dana Van Aken et al. "Automatic database management system tuning through large-scale machine learning". In: *Proceedings of the 2017 ACM international conference on management of data*. 2017, pp. 1009–1024.

[16] Ji Zhang et al. "An end-to-end automatic cloud database tuning system using deep reinforcement learning". In: *Proceedings of the 2019 International Conference on Management of Data*. 2019, pp. 415–432.

[17] Zhe Zhang. *An Automatic Source Code Generation Tool for OLTP Database Benchmarks*. 2018.

[18] Conghuan Zheng, Zuohua Ding, and Jueliang Hu. "Self-tuning performance of database systems with neural network". In: *International Conference on Intelligent Computing*. Springer. 2014, pp. 1–12.
[19] Fuzhen Zhuang et al. "A comprehensive survey on transfer learning". In: *Proceedings of the IEEE* 109.1 (2020), pp. 43–76.

A Appendix

CS310 Project Specification: Transfer learning for OLTP database tuning

Valentin Kodderitzsch (u1931737)

October 2022

Contents

1 Problem Statement

The aim of this project is to add to the current transfer learning (TL) research landscape in the domain of database management systems (DBMS) tuning via machine learning (ML) models [3]. We are concerned with the distillation and knowledge transfer from one optimized ML model to a second ML model [2].

Both models try to find the most competitive DBMS settings for a predefined, static DBMS workload. A workload is just a collection of CRUD[1] statements run against a database [4]. In our case we are concerned about Online Transaction Processing (OLTP) database tuning. The crux of our problem is that the two workloads are different. In short, the goal is to recommend the most competitive DBMS settings for the second workload by leveraging the tuning knowledge encapsulated in the first, already optimized model to avoid training the second model from scratch.

2 Research

Before exploring why the computer science community is interested in DBMS tuning it worth wile to clarify what exactly DBMS tuning is. At its core DBMS tuning is about achieving the best performance metric after some form of a stress test is executed against the database for different DBMS settings. The most common performance metrics are the DBMS throughput or latency. The type of stress test depends on the type of database. However, the most common test frameworks are TCP-C, YCSB and TCP-H [4]. When it comes to DBMS settings they are usually divided into developer level and admin level settings. Developer level settings are mostly concerned with query plans, i.e. the exact ordering of joins for instance. Admin level settings include physical design changes such as manipulating indexes, hardware resource changes and database knob setting changes. Commonly manipulated knob settings are the log file size or the buffer pool size as well as caching policies. All admin level settings affect the overall performance of the DBMS. The problem of tuning knob settings is especially interesting as it is an $\mathcal{NP}$-hard problem [4]. As such leveraging ML models to approximate the problem becomes a promising proposition for the computer science community to explore.

There are 2 main approaches to DBMS tuning as outlined by the authors of [3]. Namely, internal vs external DBMS tuning agents. An agent is usually a bigger software component that can implement changes on its own. Compared to an internal agent the external agent views the DBMS as a "black-box". This means that the agent only receives data about the DBMS via a third party application such as JDBC[2]. The model has no "knowledge" about the inner workings of the DBMS. The internal agents on the other hand is part of the DBMS architecture and can has access to more information. As such internal agents are more "knowledgeable". However, they are also more difficult to implement. Examples of external agents are BTSTR [1], OtterTune [4] and ResTune [5]. The authors of [3] also mentioned NoisePage as an example for internal tuning.

Within the context of DBMS tuning there is a research area which is interested in transfer learning. TL as described by the authors of [2], albeit in a different

[1] Create, read, update, delete
[2] A Java API to communicate with databases.

context, is the process of distilling and transferring the knowledge from a first model to a second model. This second model aims to solve a similar task compared to the first model. One of the main goals of TL is to learn the second task faster than it took to learn the first task. Ideally, the second model should also perform better at the second task compared to the first model and the first task. An analogy described by the authors of [6] is learning how to ride a motorcycle after you have learned how to ride a bicycle. Additionally, there are 2 more variations to TL. The first one is where you "unlearn" the first task, i.e. you learned how to ride a motorcycle very quickly, however, you have now forgotten how to ride a bicycle. The second variation is that the model is now able to solve both task, i.e. you can ride a bicycle and a motorcycle in the end. Both variations have different constraints and implementation challenges.

One paper which is concerned with TL in the context of DBMS tuning has been co-authored by Meghdad Kurmanji[3]. It is titled DDUp[4] but has not been published yet. The main concern of the DDUp paper is to transfer learning from one model to another when the distribution of our data set changes. In the context of a DBMS this means that new records have been inserted but we are still concerned about performance metrics such as approximate query processing (AQP) time. One of the main differences between our problem statement and DDUp is that we are concerned with workload changes instead of just data distribution changes, i.e. all CRUD statements compared to insert statements only. As such we will be working on OLTP databases whereas DDUp focused on OLAP databases. Additionally, we are specifically interested in finding the most competitive database knob settings. We are also not concerned with maintaining the original model whereas DDUp tries to avoid "unlearning".

In conclusion, the exact gap we are trying to fill in the current research landscape is how to transfer learning from one workload optimization ML model to another model when the workload changes. Even though papers such as OtterTune [4] or BTSTR [1] have touched upon TL or "workload mapping" their efforts have been focused on creating an end-to-end automated DBMS tuning agents. Our efforts on the other hand are solely focused on TL in the context of OLTP database tuning and will as such provide novel insights for the computer science community.

3 Thesis Objective

As opposed to the projects from authors of BTSTR [1] and OtterTune [4] we do not aim to create an updated end-to-end automated DBMS tuning system. Instead the goal of this thesis is to propose a novel TL approach in the domain of OLTP workload optimization. As such we want to focus our development efforts on the workload optimization and the TL modules instead of an entire

[3]PhD Candidate at the University of Warwick

[4]Detect, Distill and Update: Learned DB Systems Facing Out of Distribution Data

DBMS tuning pipeline. Those 2 modules will then be thoroughly tested with special emphasis on the TL one.

3.1 Workload optimization

To achieve the workload optimization milestone the following steps need to be completed.

1. Set up the DBMS environment
2. Find a workload test framework
3. Identify the best ML model which also allows for TL
4. Choose a dimensionality reduction technique for the DB settings
5. Create a ML model to find optimal DB setting for a specific workload A

For this milestone we can make use of the research presented by the authors of OtterTune [4], BTSTR [1] and possibly Restune [5].

3.2 Transfer Learning

For the transfer learning milestone to be achieved the following steps need to be completed.

1. Identify the best TL approach
2. Create a second ML model with distilled knowledge from the first model
3. Optimize the second model for a new workload B

Here we can make use of the ideas presented in the 2 papers co-authored by Meghdad Kurmanji[5][6] which are yet to be published. Papers [2, 6] provide a general overview of the TL landscape. Additionally, it is worth mentioning that we don't want our TL module to maintain the original model as described in Meghdad Kurmanji's paper. We only care about providing the most competitive DB settings for the new workload B.

3.3 Testing

To test our work we can follow the steps outlined in the BTSTR [1] paper. This means running a target workload against a set database server to check the workloads target metrics (e.g. throughput, latency). Additionally, it would make sense to run the following 2 TL specific tests.

1. Compare the tuning time for workload B with TL vs the tuning time of workload B without TL

[5]Detect, Distill and Update: Learned DB Systems Facing Out of Distribution Data
[6]The brainy student: Scalable deep unlearning by selectively disobeying the teacher

2. Compare the tuning performance of the model with TL vs the tuning performance of the model without TL

3.4 Possible extensions

A possible extension of the project could be to adopt out TL insights to evolving DB deployments such as changing hardware profiles. However, this step will only be reached once the previously mentioned steps have been successfully completed.

4 Methods

As mentioned in the Thesis Objective section the main approach to implementing this project is to adopt and further develop parts of our project from previous work in the area. More specifically, we will base the workload optimization module on the work presented in BTSTR [1]. Our TL module will be based on DDUp. Tesing will be based on both papers. Additional papers will be considered as needed. The authors of BTSTR and DDUp are both Warwick students which means that I will be able to reach out to them to ask for guidance. Details on how the project will be guided to completion can be found in the project management section.

4.1 Resources

In terms of resources we can again make use of the approaches presented by the authors of BTSTR and DDUp. The exact list will most likely change throughout the course of this project. However, the following list should give a good overview.

1. Access to a database server running a DBMS such as PostgreSQL [1]
2. Access to a DBMS test framework such as TCP-C [1]
3. A selection of ML libraries such as scikit-learn or TensorFlow

For item 1) and 2) I should be able to consult with supervisor and the author of [1] to figure out the details. For item 3) I will reach out to the authors of DDup and BTSTR to find out the exact libraries and model choices I should make for this project.

5 Risk Assessment

The main risk of this project is either running out of time or realizing that the original approach was not valid. As such it is important to have regular meetings to discuss the pace and trajectory of this project. A provisional meeting schedule will be laid out in the project management section.

The second biggest risk might be difficulties accessing resources 1) and 2). However, as the author of BTSTR had to access the same resources I am confident that it won't be an issue for this project either.

6 Project Management

The following section outlines how I intend on carrying out this project. The last but most important deadline is May 2nd 2023 which is when the final report is due. Before that there are 3 more deadlines to help with the progress of the project. The first deadline is due on October 13th 2022 which is this specification report. The next deadline is the progress report which is due about 6.5 weeks later on November 28th 2022. The last pre final report deadline is due between March 6th and 17th, depending on your assigned presentation date. This means that by March I need to have produced tangible results in terms of functioning code for my assessors and peers. After the presentation I will have about 2 months to convert my notes, submissions and code into a final report. This project will take about 7 months from initiation to completion.

The 4 key submissions dates are captured in the following table.

Item	**Date**	**Goal**
Specification Report	Oct 13 2022 (T1W2)	Scope problem statement
Progress Report	Nov 28 2022 (T1W9)	Start coding
Presentation	Mar 6 to 17 2023 (T2W9-10)	Complete coding
Final Report	May 2 2023 (T3W2)	Write up

6.1 Timeline

The following table aims to provide more detail about the overall project timeline. It will be reviewed regularly throughout the course of the project.

Term	Week	Goal
1	1	Project initiation
1	2	Project specification submission
1	3	Explore existing literature
1	4	
1	5	Learn about DBMS tuning implementation details
1	6	
1	7	Learn about TL implementation details
1	8	Prepare progress report
1	9	Progress report submission
1	10	Learn about TL implementation details
Winter break		Final report write up based on term 1
2	1	Implement workload optimization module
2	2	
2	3	
2	4	Implement transfer learning module
2	5	
2	6	
2	7	
2	8	Testing
2	9	Testing and presentation preperation
2	10	Project presentation
Spring break		Final report write up
3	1	Final report write up
3	2	Final report submission

Figure 1: Project timeline overview

6.2 Guidance

As mentioned in previous sections the success of this project depends on keeping the right trajectory and pace over the course of the next 7 months. Therefore,

it is essential that I am in regular contact with my supervisor and the authors of the DDUp and BTSTR papers.

I intend on speaking with my supervisor at least once a month. This will allow me to present bigger chunks of work I have completed. As such our discussions will be focused more on the general trajectory and research areas I should focus on.

I am hoping to speak to the authors of DDUp or BTSTR every 2 weeks. This will of course depend on their schedule. My goal is to discuss ideas presented in their papers with them. Additionally, I am looking for concrete guidance in terms of implementation details. For instance which ML models or libraries to choose. I am also hoping to learn more about my project through hearing about their experiences. Finally, I am hoping that having a meeting every 2 weeks will help me maintain a reasonable pace.

6.3 Documentation

This project will be documented using Google Docs and Google Drive. Throughout the project I will keep a weekly log of all the task I achieved during the week plus comments. This will hopefully make the progress and final report easier to write. Additionally, I will record any meeting notes on Google Docs. I will also continue maintaining my reading list. This list is stored on my Google Drive and is separated into "read" and "to be read" folders. All papers in the "read" folder contain a summary such that they can easily be referred back to in the future.

6.4 Version Control

This project will use Git for version control and the remote repositories will be stored on GitHub.

7 Legal, social, ethical and professional issues

There are no social and ethical[7] issues to consider with this project as no work with other people (e.g. interviews) is required.

As the project aims to add to the current TL landscaping using open-source software resources there are also no legal or professional issues to consider.

[7] https://warwick.ac.uk/fac/sci/dcs/teaching/material/cs310/ethics

8 Bibliography

References

[1] George-Octavian Barbulescu and Peter Triantafillou. "Anatomy of Learned Database Tuning with Bayesian Optimization". In: *2022 IEEE 38th International Conference on Data Engineering Workshops (ICDEW)*. IEEE. 2022, pp. 9–15.

[2] Geoffrey Hinton, Oriol Vinyals, Jeff Dean, et al. "Distilling the knowledge in a neural network". In: *arXiv preprint arXiv:1503.02531* 2.7 (2015).

[3] Andrew Pavlo et al. "External vs. internal: an essay on machine learning agents for autonomous database management systems". In: *IEEE bulletin* 42.2 (2019).

[4] Dana Van Aken et al. "Automatic database management system tuning through large-scale machine learning". In: *Proceedings of the 2017 ACM international conference on management of data*. 2017, pp. 1009–1024.

[5] Xinyi Zhang et al. "Restune: Resource oriented tuning boosted by meta-learning for cloud databases". In: *Proceedings of the 2021 International Conference on Management of Data*. 2021, pp. 2102–2114.

[6] Fuzhen Zhuang et al. "A comprehensive survey on transfer learning". In: *Proceedings of the IEEE* 109.1 (2020), pp. 43–76.