Master’s Thesis

Otto-Tuner: Automated Configuration tuning for HTAP and Multi-Objective workloads

Author:
Meghana Ravindranath Deshmukh

August 19, 2021

Advisors:

M.Sc. Gabriel Campero Durand,
Prof. Dr. rer. nat. habil. Gunter Saake

Databases and Software Engineering Workgroup,
University of Magdeburg
Deshmukh, Meghana Ravindranath:

*Otto-Tuner: Automated Configuration tuning for HTAP and Multi-Objective workloads*

Abstract

It is known that, with the advent of technology, the amount of data produced, stored and used has increased drastically. Not only the size of the data has increased, but the types of data or workloads have also diversified. With these changes, it becomes necessary for databases to support different types of large data and its management. To provide the best support, such databases needs to be tuned to each data and workload type. The tuning of the database refers to assigning the right value for each configuration of the database settings. This tuning of the database settings forms the backbone of database performance. Unfortunately, the number of tunable knobs grows with each database generation, the knobs might have non-trivial relationships between them, and the knowledge from tuning one database system might not easily map to another. To aid this process of configuration tuning, Artificial Intelligence has been recently proposed to partially automate the process.

Today, for automated configuration tuning, some of AI models are used. Support for the automated tuning is provided to only some of commonly used workloads. This thesis evaluates and compares each of the AI models on these workloads to find, among a scoped choice, the algorithm that could perform better. The performance of the models which use sampling techniques are highly influenced by the choice of sampling techniques, which we also propose to study.

In addition, the extension of the support is needed for hybrid workloads and multi-objective optimization techniques are called upon for tuning systems. For the optimization of more than one objectives, this thesis considers scalarization and individual optimization of objectives. Among the two methods, multi-objective optimization techniques proved to fare well.
Acknowledgements

It has been a cherish-able and most knowledgeable experience during the course of this thesis. I wish to convey my gratitude to every person who made that possible.

First and foremost, I would like to thank M.Sc. Gabriel Campero Durand, my supervisor. In every step during the course of this thesis, I have received incredible amount of guidance and support from him. He has been highly motivating and inspiring when faced with hurdles during our work together. I thank him for everything that he has made possible in this thesis.

I would like to extend my sincere regards to Prof. Dr. rer. nat. habil. Gunter Saake for providing the opportunity to work on my thesis, under his chair.

I would like to thank Dr.-Ing. Christoph Steup for being the second reviewer of this thesis. I am immensely grateful to him for obliging to provide his knowledgeable insights on this thesis.

Also, I would like to thank my friend Yash Shah for his contributions in bringing up the experimental setup.

Finally, I would like to thank my closest family and friends who have been immensely understanding and providing unbounded support throughout my master’s tenure. Thank You.

I would like to mention that with the end of this thesis, my collaborative relationship with Otto-von-Guericke-Universität, Magdeburg is reached its end. It has been a memorable experience working along with the DBSE group.
Declaration of Academic Integrity

I hereby declare that I am the sole author of this master thesis and that I have not used any sources other than those listed in the bibliography and identified as references.

Magdeburg, August 19th, 2021

Meghana Ravindranath Deshmukh
# Contents

List of Figures xvi

List of Tables xviii

1 Introduction 1
  1.1 Motivation: ........................................... 2
  1.2 Main Contributions: .................................... 3
  1.3 Thesis structure: ....................................... 6

2 Background 9
  2.1 Overview .............................................. 9
  2.2 AI for Database Configuration Tuning ..................... 11
    2.2.1 Tuning of all configurations ....................... 12
    2.2.2 Single resource tuning ............................ 22
    2.2.3 Field Study: ...................................... 24
  2.3 AI models used for configuration tuning .................. 25
    2.3.1 Reinforcement Learning ........................... 26
    2.3.2 Deep Deterministic Policy Gradient (DDPG) ......... 28
    2.3.3 Maximum a Posteriori Policy Optimisation (MPO) .... 29
  2.4 Transactions vs Analytics .............................. 30
    2.4.1 TPC-C ........................................... 30
    2.4.2 TPC-H ........................................... 32
  2.5 Open directions in AI for configuration tuning .......... 34
  2.6 Summary: ............................................. 36

3 Architectural Overview and Research Questions 37
  3.1 Research Questions .................................... 37
  3.2 System Architecture ................................... 38
    3.2.1 Architecture of Otto-Tuner ....................... 39
    3.2.2 Transformation of OtterTune to Otto-Tuner .......... 40
List of Figures

2.1 Working of BestConfig in a 2D search space [ZLG+17] . 13
2.2 Architecture of BestConfig [ZLG+17] . . . . . . . . . . 14
2.3 Architecture of CDBTune [ZLZ+19] . . . . . . . . . . 15
2.4 Architecture of QTune [LZLG19] . . . . . . . . . . 16
2.5 Working pipeline of QTune [LZLG19] . . . . . . . . . 16
2.6 Working of iTuned planner [DTB09] . . . . . . . . . . 18
2.7 iTuned architecture [DTB09] . . . . . . . . . . . . . . 19
2.8 Multi-Model offline training flow [GYSR21] . . . . 20
2.9 Multi-Model tuner overview [GYSR21] . . . . . . . . 20
2.10 Cycle of reinforcement learning [RLC] . . . . . . . . 27
2.11 Classification of Deep Reinforcement Learning Algo-
rithms [LCP+21] . . . . . . . . . . . . . . . . . . . . 28
2.12 OLTP and OLAP workloads comparison [ALL19] . . 32
3.1 OtterTune Architecture [VAPGZ17] . . . . . . . . . . 39
3.2 Otto-Tuner (Extended OtterTune) Architecture: De-
picting the five major changes made to the original ot-
tertune [VAPGZ17] architecture. . . . . . . . . . . . . . 40
3.3 Knob Attributes Example . . . . . . . . . . . . . . . . . 42
3.4 Metric Attributes Example . . . . . . . . . . . . . . . . 42
3.5 Information Flow in OtterTune . . . . . . . . . . . . 47
3.6 Information Flow in OtterTune for MPO agent . . . . 49
4.1 postgreSQL Architecture Overview [Opr19] . . . . . . . 54
4.2 SingleStore (memSQL) Overview [Sin19] . . . . . . . . 56
4.3 HTAP description [Har16] . . . . . . . . . . . . . . . . 58
4.4 Multi-Objective optimization by MO-MPO [AHH+20] . 62
4.5 MO-MPO policy iteration algorithm. [AHH+20] . . . . 64

5.1 DNN model performance on tuning 25 Postgres knobs for OLTP workload . . . . . . . . . . . . . . . . . . . . . . 74
5.2 DNN model performance on tuning 25 SingleStore knobs for OLTP workload . . . . . . . . . . . . . . . . . . . . . . 75
5.3 Comparison of model performance on Postgres DBMS . 77
5.4 Comparison of model performance on SingleStore (memSQL) DBMS . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 77
5.5 GPR execution with random sampling on OLAP workload and Postgres DBMS . . . . . . . . . . . . . . . . . . . . . 77
5.6 GPR execution with random sampling on OLAP workload and SingleStore DBMS . . . . . . . . . . . . . . . . . . . . . . 80
5.7 Comparison of sampling techniques on Postgres DBMS 81
5.8 Comparison of sampling techniques on SingleStore (memSQL) DBMS . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 82
5.9 Comparison of scalarization techniques . . . . . . . . . 85
5.10 Execution of HTAP workload on SingleStore DBMS using simple multiplication scalarization . . . . . . . . . . . . . . . 86
5.11 Execution of HTAP workload on Postgres DBMS using simple multiplication scalarization . . . . . . . . . . . . . . . 86
5.12 Comparison of scalarization with Multi-Objective Optimization . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 90
5.13 Execution of Mo-MPO on SingleStore DBMS for HTAP workload . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 91
5.14 Execution of Mo-MPO on Postgres DBMS for HTAP workload . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 91
List of Tables

2.1 Summary of Configuration Tuning solutions ............... 22
2.2 TPC-C workload benchmark parameters [TPC16a] ....... 32
2.3 TPC-H workload benchmark parameters [TPC16b] ........ 33

3.1 Knob Attributes as stored in Otto-Tuner ................. 43
3.2 Metric Attributes as stored in Otto-Tuner ............... 43

4.1 HTAP workload benchmark parameters [CPV+17] ....... 58
4.2 Hardware component characteristics of the experiment bed. ......................................................... 67

5.1 Workload characteristics for experiments ................. 71
5.2 Experiments for evaluation of tuning models. ............ 74
5.3 Average Throughput gain based on model and database 76
5.4 Experiments to compare sampling techniques ............ 79
5.5 Experiments to compare scalarization techniques ....... 84
5.6 Experiments to evaluate multi-objective optimization .. 89

A.1 Chosen knobs to tune for PostgreSQL .................... 100
A.2 Chosen knobs to tune for memSQL ....................... 101
A.3 TPC-C workload bench parameter values for Single-Store (memSQL) ........................................ 102
A.4 TPC-C workload bench parameter values for Postgres .. 102
A.5 TPC-H workload bench parameter values for Single-Store (memSQL) ........................................ 102
A.6 TPC-H workload bench parameter values for Postgres . 102
A.7 HTAP workload bench parameter values for SingleStore (memSQL) . . . . . . . . . . . . . . . . . . . . . . . . . 103
A.8 HTAP workload bench parameter values for Postgres . 103
1. Introduction

Database management systems (DBMSs), often called just databases, are widely used today as a storage utility in many business applications. As the usage of DBMSs has increased, more focus is drawn towards improving the performance of the systems. A closer study of the DBMS performance suggests a tight relationship with the DBMS configuration and settings [VAPGZ17]. The right DBMS settings can make the system display improved performance for a workload. This direct dependence of the DBMS settings on performance stresses on the role of the database administrator (DBA), who is commonly responsible for finding the best performance settings.

DBAs tune the settings of the DB (databases) to match the requirements of the system which is using the DB, often based on personal experience. Even though this seems like a solved problem, due to the wide range of available databases and its settings, this very process is error prone. There are multiple databases available in the market for most of the specific requirements. For each of the these databases, the settings or configuration knobs might differ with name, unit or scope. This drastically widens the knowledge space for the DBA. Apart from the wide range of setting possibilities, each of the configuration knobs have different and complex relationship with the required target objective. This makes the job of the DBA tough and highly error prone. To avoid errors and maintain consistency, the DBA is provided help with the recourse to automated tuning of the database settings.
This thesis works towards improving the process of configuration tuning. This chapter introduces the reader to the premise of configuration tuning, motivations for the contributions in this thesis and the structure of the thesis. This chapter is structured as follows:

- **Motivation:** Section 1.1 provides an overview of the contributions made in the configuration tuning field till date. This section further goes on to highlight the possible improvements and the reasons that make these improvements important, serving as motivation for our research.

- **Main Contributions:** Section 1.2 highlights the focus of the thesis. This section also briefs about the stakes that has led us to seek to contribute in the direction.

- **Thesis Structure:** Section 1.3 provides an overview of the structure of the thesis.

### 1.1 Motivation:

Database settings are tightly coupled with the workload on the database. For the database to perform its best at a workload, the each of the configuration knob should be tuned to its optimal value. Each knob’s optimal value depends both on the workload and the optimal performance (represented by target objective) of the database.

So, What are these workloads? Workload represents the pattern of operations (SQL commands) executed on the database for the system using the database to function. The main categories of workloads comprises of OLAP (Online Analytical Processing), OLTP (Online Transaction Processing) and HTAP (Hybrid Transaction / Analytical Processing). OLAP and OLTP workloads are the classic workload types among which there exists a clear distinction. Whereas, HTAP represents the hybrid version of the classic workloads. To get a clear view, consider a spectrum where one end represents OLAP and the other end represents OLTP. In between these ends stand HTAP workloads. These workloads pose challenges as they must balance the competing needs of OLAP workloads that need a lot of parallelism within a query, and OLTP that need synchronization between the database components and many small threads.
1.2 Main Contributions:

There exists quite a few automated tuning solutions to aid the DBA’s role ([ZLZ+19], [KB20], [ZLG+17], [DTB09], [YG], [LZLG19], [VAPGZ17]). These solutions mainly target on the OLAP and OLTP workload types. As these types have clear distinction between them, a given workload can be categorised into either OLAP and OLTP with lesser difficulty. The third workload type HTAP [Har16] is a combination of both OLTP and OLAP. To improve HTAP performance, it is necessary to improve on both the objectives of both OLTP and OLAP. Existing automation techniques focus on improving one objective at a time. Hence, the these solutions work on HTAP workloads but rather poorly.

As with the advent of new technology, the third workload type HTAP is being extensively used. HTAP (Hybrid Transactional and Analytical Processing) provides the required trade off between the old and processed data and most recent and less structured data. This enables the businesses today to draw conclusions and make business decisions based on the recent as well as historical data. Based on the requirement of each business or the system using the database, the trade off differs. This implies that, each HTAP workload differs with the degree of OLAP and OLTP transactions being combined in the workload. This helps us understand that not every HTAP workload is the same and not all the HTAP configuration gives the optimal performance for the database.

Even though HTAP workloads are being widely used today, this is a relatively new workload type. This also applies to the automation of the configuration tuning process done by the DBA. Both being the relatively new field of science and technology, there exists a plethora of unsolved problems. Automated configuration tuning for HTAP is one of the problems that need addressing.

1.2 Main Contributions:

The focus of this thesis study is towards analysis and improvement of configuration tuning for database systems. This section lists the main contributions made in this thesis towards researching the field of configuration tuning.

- **Thorough comparison of AI techniques for tuning:** Configuration tuning of database knobs is a mundane task that needs
Introduction

Multiple repetitions by the DBA. As this process is much error prone and tedious to a human, it needs to be automated. Some of the well performing automated configuration tuning solutions are: [ZLZ+19], [KB20], [ZLG+17], [DTB09], [YG], [LZLG19], [VAPGZ17]. These solutions have used different methods to automate the tuning process. It is important to know which among these are better performing and supports further improvement in the system. With this knowledge, further research can be done to generate better tuning systems.

The knowledge of the better performing technology helps harvest the same to yield better results. In other words, knowing which technology or AI model performs well paves paths for better solutions in research. With that in mind, this thesis analyses the major categories of methods used in tuning today. Using this analysis, the best performing model among the tested will be compared against the new contributions made in this thesis. The main idea behind this contribution of thorough analysis in the field is to steer the direction of the research towards the best performing tuning model.

- Configuration tuning for hybrid workloads: Most of the tuning solutions ([ZLZ+19], [KB20], [ZLG+17], [DTB09], [YG], [LZLG19], and [VAPGZ17]) today, focuses on tuning mainly two types of workloads. Even though some of the solutions consider varying workloads, the variation is focused on the changes within the same class of workloads.

With the advent of small scale businesses, the new hybrid workload [Har16] is most widely used. To support these workloads, databases are providing support. But is there support from the tuning perspective towards hybrid workloads? To the best of my knowledge and the research done, there are no tuning solutions that support hybrid workloads.

This thesis focuses on providing support for configuration tuning of hybrid workloads. This thesis uses the available benchmarking tool for hybrid workloads [CPV+17]. Using this benchmark, workloads are generated and we test the database against the performance of the required target objective.
1.2. Main Contributions:

- **Multiple objective tuning in single objective systems.** As mentioned earlier, this thesis focuses on proving configuration tuning support for hybrid databases and workloads. The hybrid workload measures the performance of the database against the benchmark [CPV+17]. The metric used to measure the performance of the database against the workload is Unified HTAP metric (UHTAP) [Paw19]. The Unified HTAP metric [Paw19] which indicates the performance of database against HTAP workloads is a multi-objective metric. In other words, the metric doesn’t give a single value rather two values which depicts the performance of two different features. In order to integrate this hybrid workload tuning in the existing tuning solutions, the unified HTAP metric needs to be modified to depict a single value.

The existing solutions for tuning today supports single objective tuning. In order to integrate the tuning of hybrid workloads, some of the scalarization techniques ([Bra18], [Gun18]) which transform the multiple objective result into a single objective results are used. This thesis shows the work done to integrate the hybrid workload evaluation metric into the single objective tuning system using the scalarization techniques. This thesis further goes on to compare which of these scalarization technique helped achieve the best performance.

- **Configuration tuning systems with multiple objectives:** This thesis not only focuses on analysing the existing results but also to make progress in the research of the configuration tuning automation. In order to do so, this thesis introduces hybrid workload tuning. Also tries to integrate hybrid workload tuning into the existing system solution.

One such attempt to integrate hybrid workload tuning to existing systems is the use of scalarization techniques. Unfortunately, scalarization comes with the possible loss of information. Hence, the desired approach would be to not transform a result rather to use it as a whole and still be able to tune the configurations.

In order to reduce the information loss in transformation and still be able to integrate with existing solutions, this thesis explores the field of multi-objective tuning agents [AHH+20]. These mul-
Multiple objective tuning agents accepts more than one target objectives or the result values. Every objective’s value is used a separate reward and thus minute variations in each of the objectives are captured. This thesis contributes to the research by implementing this multi-objective agents for configuration tuning.

1.3 Thesis structure:

In order to work on the contributions made in this thesis, it was necessary to learn the existing research and the contributions already made in the field. Section 1.3 provides a detailed view of the research materials used to understand the contributions prior to this thesis in the field. This chapter helps the reader understand the knowledge for the work done in this thesis. It also provides a background for understanding the contributions made in this thesis. Section 1.3 studies the trend of configuration tuning automation and the possible paths to explore. Section 2.2 draws a picture of the progress made by fellow researchers in the field of configuration tuning automation for database knobs. Section 2.3 explains the various artificial intelligence models used to automate configuration tuning. Section 2.4 provides insights on the major workload types. Section 2.5 highlights unexplored areas within configuration tuning. This also highlights areas towards which this thesis will be contributing.

One of the existing tuning solutions [VAPGZ17] is used as a base for this thesis. This solution is further modified to support all the contributions made in this thesis. Chapter 3 provides an insight into the architecture of this tuning system. This chapter also highlights the changes made on the system to support the contributions of this thesis. Section 3.1 lists the research questions which help to structure this research. Section 3.2 helps to understand the changes made to the existing architecture of [VAPGZ17]. Section 3.3 draws insight on the changes made to accommodate the hybrid workloads, benchmarks and metrics into the tuning system.

To conduct the experiments for this thesis, a experimental system setup was created. The details about the setup are provided in Chapter 4. Section 4.1 explains in detail about the databases and workloads that were modified and changed for this thesis. The changes which
support multiple objective optimization are detailed in Section 4.2. Section 4.3 provides the software and hardware characteristics of the system on which the experiments of this thesis are conducted.

We evaluate the experiments of this thesis in Chapter 5.

This thesis opens doors to many possible opportunities in the field of automation tuning and its optimization. Chapter 6 provides a summary of this thesis work and highlights possible future work in the area.

Some of the concepts or aspects that were used and less illustrated or explained are detailed in Chapter A.
1. Introduction
2. Background

This chapter provides the reader an understanding of the research focus of this thesis. With the aim to provide a context for the work done in this thesis, the chapter is structured as follows:

- **Overview:** Section 2.1 briefs about the sources and research papers that were used in the study of this thesis.

- **Artificial Intelligence for Database Configuration Tuning:** Section 2.2 provides the background of current research in the field of databases in terms of automation of the configuration tuning task.

- **AI models used for configuration tuning:** Section 2.3 explains and provides a background for the Artificial Intelligence models that are used in the thesis.

- **Transactions vs Analytics:** Section 2.4 provides an overview of common database workloads, types and their relevance for configuration tuning. This section focuses on the workload types used most commonly in configuration tuning research till date.

- **Open Directions in AI for Configuration Tuning:** Section 2.5 discusses the possible areas for innovation in database and configuration tuning and optimization.

## 2.1 Overview

The databases are in existence since 1960s. Today, databases are the most widespread and in demand. With the increase in usage
of the databases, the requirement for more diverse, faster and efficient databases have increased. For these requirements to be met, the maintenance and tuning of the databases play the most important role. Traditionally, these tasks are done by a human DBA (Database administrator). The time taken by a human DBA to optimally configure and tune the database ranges from days to weeks. Also, with the huge number of configuration knobs and metrics, it can be daunting for a human DBA to find the globally optimal solution all the time.

To overcome these problems, the research is oriented towards applying Artificial Intelligence (AI) to automate the tasks associated with DBMS. AI helps automate the complex and repetitive tasks that required DBA’s expertise. It also helps minimise the error and time taken to achieve optimum performance of the DBMS.

This thesis aims at making progress towards artificial intelligence (AI) in databases (DB). With this in mind, our research review mainly focused on the following.

- Presenting and introducing the current research in AI for DB.
- Background and study of the commonly used AI models in DB
- Open directions for growth of AI in configuration tuning

The survey [ZCLS20] was used as the starting point for the study of research in the AI for DB. The [ZCLS20] categorises the research work in terms of the areas in which AI is applied in DBMS. Database design, configuration tuning, query optimization, resource optimization, database monitoring and its security are the major areas of AI application in DBMS today.

Among the broad range of categories and the applications of AI in DB, the one that classifies the contribution of this thesis is automated configuration tuning. For understanding the current state of research in terms of automated configuration tuning, the most relevant was [VAPGZ17]. [VAPGZ17] is one among a few of the automated tuning algorithms. The reasons to choose [VAPGZ17] and the classification of other tuning algorithms are further explained in detail in Section 2.2.

Having understood the need of automation in DBMSes, it is also necessary to know how it is automated. The study of models which are used
to automate the repetitive, time consuming and most difficult tasks of
DBMS, provides clarity of how automation helps provide better op-
timization. Among the vast algorithms of AI, very few are actually
used in the applications of DBMS automation.

One such algorithm is Deep Deterministic Policy Gradient (DDPG)
\cite{LHP+19}. DDPG is the most used AI algorithm that has provided sig-
nificant improvement in terms of performance for DBMS. Even though
DDPG is efficient and widely used, it fails to support multi objec-
tive optimization. To help optimise multiple objectives, the algorithm
Maximum a posteriori Policy Optimization (MPO) \cite{AST+18} can be
extended. The extended version of MPO which supports multiple ob-
jectives is Multi-Objective MPO (MO-MPO) \cite{AHH+20}. These algo-
rithms are further explained in more details in the sections Section 2.3
and Section 4.2.3 respectively.

AI for DB is relatively new field with booming research interests. This
has left many more ideas and possible open path unexplored. This
thesis attempts to contribute to one such path. Research and study
was also made to aid the understanding of the new possibilities in the
field. The materials used are further explained in detail in Section 2.5

\section{2.2 AI for Database Configuration Tuning}

Database Management systems comes with a hundreds of configura-
tions. These configurations differ from the database to database and
the value it takes differs from one hardware setting to another. In
spite of its difficulty to handle, these configurations play a vital role
in the DBMS’s performance. This huge impact towards the working
of the DBMS, enforces to prioritise configuration tuning to improve
DBMS performance.

Traditionally, Database Administrator (DBA) manages and configures
the configurations knobs of the DBMS. DBA also needs to configure
different settings for the knobs for different workload requirements.
For the DBA to be successful at configuration tuning and achieve the
known optimal result, it takes weeks. It also requires the DBA to be
aware of the inter-dependencies between the knobs, and the relation-
ship between the knobs and the metrics that are being configured. This
puts massive pressure and dependence on the DBA alone to achieve
optimal performance of the DBMS. Which in turn leads to errors and sub optimal performance.

In order to aid the DBA, the configuration tuning process is being automated with the help of AI. Today, there exists nearly a dozen of algorithms and systems which aid the tuning process and effectively improve the performance. Many research techniques have contributed to the field and has helped us reach the place we are today. Some of the many such significant algorithms and techniques that have contributed in the field, and has relevance to this thesis are listed and summarised below in Section 2.2.1 and Section 2.2.2.

### 2.2.1 Tuning of all configurations

This section focuses on the solutions which enable tuning of all the configurations in the database. This thesis focuses mainly on tuning all the tunable configurations of the database. This causes the search space for the optimal configuration to be significantly larger. The research papers mentioned in this section fall under the category of tuning all tunable configurations.

- **Bestconfig: Tapping the performance potential of systems via automatic configuration tuning. [ZLG+17]**: The paper proposes a system *BestConfig* which helps in tuning the configurations of general systems. With the proposed *divide-and-diverge sampling* methods and *recursive-bound-and-search method*, the author evaluates the performance of *BestConfig* for cloud based databases.

  The main contribution of the paper constitutes *divide-and-diverge sampling* methods and *recursive-bound-and-search method*. *BestConfig* aims to tune the database system even under the crunch of resources. The sampling technique *divide-and-diverge sampling* helps to achieve the same. *divide-and-diverge sampling* (DDS) sampling, divides the value ranges of all the parameters into k intervals. Taking the combination of all of these intervals would lead to exponential value sets. To avoid this, the algorithm takes the permutation of intervals instead. Hence reducing the number of intervals need to be tested to find the optimal solution. *BestConfig* further confirms that this does not lead to missing any
2.2. AI for Database Configuration Tuning

global maxima/minima. DDS also makes sure to remember already visited grid to avoid looking at the wrong place repeatedly. *recursive-bound-and-search (RBS)* algorithm in combination with DDS helps *BestConfig* balance between exploration and exploitation of the search space. RBS finds the best performing grid among the samples found from DDS sampling. Using this grid alone, RBS samples the next few samples. This allows the exploitation of the grid in search of optimal configuration. During the recursion step of the algorithm, RBS samples again from DDS sampling. This allows the complete exploration of the search space and avoids the algorithm getting stuck in a single search grid.

Figure 2.1: Working of *BestConfig* in a 2D search space [ZLG+17]

Figure 2.1 displays an illustration of working of DDS and RBS together in a 2D search space. The outer grid represents the grid obtained from sampling using DDS method. The inner bounded space represents the area being exploited using the RBS algorithm.

Figure 2.2 depicts the architecture of [ZLG+17]. In the figure, SUT refers to System Under Test.

The configuration settings generated under the specified constraints and the performance metrics generated from the observations of SUT are tuned by the performance optimiser to provide an optimal settings. The architecture of [ZLG+17] shows that the performance optimiser makes sure that the resource limit is not
exceeded. If an optimal solution is not yet obtained, the limit itself is added as another constraint for further tuning steps.

Authors of the paper [ZLG+17] have tested BestConfig on Cassandra/YCSB workload, MySQL and Tomcat database systems. The results of the test have shown an improvement in the performance of all the tested DBMSes. This improvement is relative to the default settings the DBMS comes with.

- **An end-to-end automatic cloud database tuning system using deep reinforcement learning.** [ZLZ+19]: The paper proposes an end-to-end configuration tuning system CDBTune. CDBTune focuses on cloud based database systems and utilizes the reinforcement learning tuning agent Deep Deterministic Policy Gradient (DDPG) to find the optimal configuration setting. The process of finding the optimal solution in CDBTune involves two stages: offline and online training. During offline training, CDBTune uses the workload benchmarks to generate the training data and trains the model uses the benchmark generated data and results.

This stage of offline training in CDBTune helps in finding the optimal configuration. CDBTune offline training pre-trains the model and keeps it ready for online training. This helps CDBTune to provide an effective solution with minimum number of samples being required during online training.

These training results are stored as a set of 4 (quadruples). Set of 4 includes the queries executed, set of knobs and their values, state in which the queries were executed and results of the queries.

To initiate, online training and help CDBTune find the right configuration for the database, user needs to initiate a request to
2.2. AI for Database Configuration Tuning

*CDBTune*. Once *CDBTune* receives a request to train, it collects the metrics for the databases, uses the pre-trained offline model and trains online. During the online training the number of iterations are determined by pre-configured setting. This can be overridden by user and can let the online training be either reduced or extended until the desired improvement is seen.

![Architecture of CDBTune](image)

*Figure 2.3: Architecture of CDBTune [ZLZ+19]*

*Figure 2.3* shows the architecture of *CDBTune*. As can be seen in the architecture, client or cloud initiates the tuning request which is received by the controller. This request can be generated by the cloud as online training and by the client side user as an offline training. Controller then tunes the system online or offline depending on the request initiator.

Once the request is initiated and received, the tuning system collects the metrics by the *Metrics Collector* component. This is then stored within the textitmemory pool and can be used in future by the tuning system.

The data collected by *Metrics Collector* and stored in textitmemory pool is further used by the Deep RL algorithm DDPG. DDPG after tuning based on the provided data samples, recommends an action as the next optimal configuration set. This configuration is applied on the database to test and collect the metrics. This process continues until a model is trained or until the user finds a satisfied solution.
The authors test \textit{CDBTune} against the existing tuning methods on seven different types of systems varying with respect to memory and disk size. \textit{CDBTune} is also tested against otter-tune [VAPGZ17] running on deep learning agents. These results show that \textit{CDBTune} with search based tuning performed better against the otter-tune [VAPGZ17] deep learning based agents.

- QTune [LZLG19]: A Query-Aware Database Tuning System with Deep Reinforcement Learning. 2019. : The paper proposes a reinforcement based tuning algorithm QTune which configures the knobs in three different granularity - Query level, workload level and cluster level. In order to tune in all three granularities, QTune uses reinforcement learning algorithm DDPG and its extension Double State - DDPG(DS-DDPG).

![Figure 2.4: Architecture of QTune [LZLG19]](image1)

![Figure 2.5: Working pipeline of QTune [LZLG19]](image2)

\textbf{Figure 2.4} depicts the architecture of QTune.

For Query-level tuning QTune feeds the query to \textit{Query2Vector} component. \textit{Query2Vector} calculates the cost of the query and
creates a vector based on the query properties. This vector is then fed to the *Tuner* which has the DS-DDPG model. The model trains and suggests a suitable configuration for the query. This configuration is applied to the database before running the query. As this tuning involves changes in configuration for every query, only the knobs which are session-level are tuned. The same flow is explained in the figure Figure 2.5.

Workload-level tuning is similar to that of query level tuning where in, instead of a single query, the component *Query2Vector* generates the vector for all the queries in the workload. This set of vectors are then modified to generate a single consolidated vector. Using the consolidated vector, the DS-DDPG generates a configuration of knobs Figure 2.5.

Cluster-level tuning differs to the other two tuning granularities. As in cluster, the amount of queries that needs to be processed are more, an optimization is introduced. To optimise this process, the *Query2Vector* generates a discrete value for each of the queries instead of a value in the continuous action space. This saves a great amount of tuning time and expenses. Using this generated discrete values for the knobs, the queries are clustered into groups. For each of these groups, a different configuration is generated. Different groups of queries are executed with their corresponding configuration settings once applied. Figure 2.5 Cluster level tuning groups the queries into clusters and different configurations are used for each cluster.

The authors test *QTune* using experiments conducted on three workloads - TPC-H, SysBench and JOB. The experiments are conducted on a system having 128GB RAM, 5TB disk, and 4.00G CPU. The metrics used are latency and throughput. From the experiments, the authors derive that higher the number of knobs tuning, higher the improvement seen. Another notable result is that, among all the levels of tuning, the cluster level tuning has provided best improved in throughput and query level tuning has provided best improvement in latency.

- **Tuning Database Configuration Parameters with iTuned [DTB09]:** The paper proposes iTuned algorithm. iTuned uses two major components: Planner and Executor. Using these com-
ponents, iTuned predicts the response surface distribution. With
the predicted response surface, iTuned tries to find the global
minima/maxima.

![Figure 2.6: Working of iTuned planner [DTB09]](image)

The main idea in the paper is to minimise the resource consump-
tion without affecting the performance of the algorithm and the
found optimal solution. To ensure this, the paper proposes a
new sampling method called *Expected Improvement Maximiza-
tion*. Initially, the samples are generated using *Latin Hyper-cube
Sampling* (LHS) such that the minimum distance between each
samples is maximum. Using these initial points, experiment re-
sults are noted forming the initial training data set. Initial train-
ing data set is used by the Gaussian process Representation of
a response surface. This generates a distribution which predicts
the behaviour of the response.

Figure 2.6 show the workings of the planner. The dots repre-
sented as circles denote the training data set points. The green
line denotes the predicted distribution of the response surface.
The two dots denote the confidence interval. Using this green
line, further prediction is done to choose the next set of experi-
ments to conduct. The experiment which provides the maximum
improvement for the current configuration settings are chosen.

Executor on the other hand, focuses on executing the selected
experiments on the workbench without affecting the production
environment. Figure 2.7 show that the user provides the policies
which determine the workbench (the system on which the experi-
ments are run) and the time to use the same. The executor follows
this policy and uses the workbench which is available to conduct
experiments. If no suitable time is provided, the executor probes the CPU to check the CPU usage. Once the CPU usage is below 10%, the executor goes ahead with the experiments. Which configuration settings to use is determined by the planner.

Using the planner and executor, *iTuned* predicts the response surface. Using the predicted response surface *iTuned* aims to find the configuration that performs optimally in the given workbench.

In order to test the efficiency of *iTuned*, authors run experiments on two databases: Postgres and MySQL. Workloads OLAP and OLTP are used to evaluate the performance of *iTuned*. *iTuned* performance is compared against the performance of default configurations that come with database and that of rule-based manual tuning. Against the default settings, 77% of improvement is seen over with the settings recommended by *iTuned*. Whereas the performance of rule-based manual tuning comes close to the performance of *iTuned*.

- **Adaptive Multi-Model Reinforcement Learning for Online Database Tuning** [GYSR21]: This paper proposes changes on the DDPG model to create a multi-model adaptive tuning algorithm. The algorithm uses both offline and online training to provide the most optimal solution.
Figure 2.8 shows the offline training pipeline of the system. The main focus of offline training is to have as many pre-trained models as possible. These offline trained models are stored in the *RL models repository*. This repository is matched and used to find the best configuration during online training as well as in the deployment setting. This ensures that during online training the models are available to further adjust to the changes in the online environment. The models are stored as a vector of database state metrics, throughput and latency. The values of these metrics are obtained from the auto-encoder neural network algorithm.

The Figure 2.9 shows the overview of the workings of multi-model algorithm for database tuning. When the user wishes to tune the
database, the user commands the tuner. The tuner, which is
the reinforcement algorithm DDPG, collects the metrics from the
database matches with nearest model in the RL repository. In
order to match the models in the repository, the tuner converts
the collects metrics into the vector. This vector is further match
with the RL repository models using cosine similarity. The tuner
also maintains a threshold value which needs to be crossed for the
model to be considered for further tuning and recommendation
of settings.

This paper mainly focuses on OLTP transactions during the off-
line training. whereas for online training, the authors try to
simulate the most realistic environment by continuously chang-
ing the workloads. Apart from considering the changes in the
workloads, the authors also consider the noise in the collected
data metrics. In order to simulate the noise as well, Gaussian
noise is added to the action space during exploration.

With the intend to obtain the best during each training session,
the algorithm tries to use a priority replay buffer. This priority
replay buffer is prioritised based on the training loss error. This
ensures that not information samples are not used less and avoids
derailing of the training session.

This paper also tries to use multi-objective reward function. In
order to focus on improving both throughput and latency, the au-
thors propose a customised reward function. This proposed for-
formula to calculate the throughput and latency ensures the function
is simple and the output stays a positive integer. This is a rel-
ative quantization of throughput and latency. The initial values
are used to gauge the improvements in every step.

The main focus of the paper are prioritising the replay buffer, in-
corporating the noise to simulate the real world environment, and
using the customised reward function. Using these three changes
and the online training implementation, multi-model configura-
tion tuning solution produces high improvements in the database
performance based on the experiments. The experiments con-
ducted focused on TPC-C workloads on Postgres database. The
performance was compared with that of offline tuning single mod-
els.
Table 2.1 provides a very precise comparison of the algorithms in configuration tuning today. To the best of my knowledge and the background research done, the table shows all the configuration tuning solutions present as of today. As can be seen in the table, the work in this area started in 2009 and has been consistently improved ever since. Before the usage of AI in database configuration tuning, search-based algorithms were extensively used. It is worth noting that these algorithms are using DDPG algorithm and its variations. Whereas there exists several other algorithms that work with continuous action space. Also, even though [GYSR21] uses both latency and throughput in its reward calculations, the values are combined to form a single reward value (scalarization). The values of both throughput and latency can be considered separately in multi-objective optimization problems.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Year</th>
<th>Method</th>
<th>Databases</th>
<th>Workloads</th>
<th>Max knobs tuned</th>
<th>Time to tune (hrs.)</th>
<th>Time to train (hrs.)</th>
<th>Gain (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>iTuned</td>
<td>2009</td>
<td>Search Based</td>
<td>Postgres, MySQL</td>
<td>Sysbench, YCSB, HiBench, JMeter</td>
<td>30</td>
<td>4.5</td>
<td>1-2</td>
<td>77</td>
</tr>
<tr>
<td>BestConfig</td>
<td>2017</td>
<td>Search Based</td>
<td>Cassandra, MySQL, Tomcat, Hadoop</td>
<td>YCSB, TPC-H6, TPC-C, Sysbench</td>
<td>109</td>
<td>4.16</td>
<td>1.25-1.45</td>
<td>63</td>
</tr>
<tr>
<td>QTune</td>
<td>2018-19</td>
<td>DS-DDPG (DRL)</td>
<td>Postgres, MySQL, MangoDB</td>
<td>TPC-W, TPC-H</td>
<td>60</td>
<td>&lt;0.1 (per query)</td>
<td>1</td>
<td>90</td>
</tr>
<tr>
<td>CDBTune</td>
<td>19</td>
<td>DDPG (DRL)</td>
<td>Postgres, MySQL, MangoDB</td>
<td>YCSB, TPC-C, TPC-H</td>
<td>266</td>
<td>0.4</td>
<td>2</td>
<td>66</td>
</tr>
<tr>
<td>Multi-Model RL</td>
<td>2021</td>
<td>DDPG</td>
<td>Postgres</td>
<td>Sysbench (OLTP)</td>
<td>16 (No restart)</td>
<td>-</td>
<td>-</td>
<td>200-500</td>
</tr>
</tbody>
</table>

Table 2.1: Summary of Configuration Tuning solutions

2.2.2 Single resource tuning

Apart from the solutions which consider all the configurations to tune, there exists solutions which focus on a particular resource to tune and optimise. These solutions also bring notable improvement in the
database performances. These solutions may not fall under the same category of the thesis but are similar in terms of the goal achieved (database performance optimization). This section provides an overview of the single resource tuning solutions.

- **Black or white? how to develop an autotuner for memory-based analytics** [KB20]: The paper highlights the possible levels in which memory management can be achieved: *application, container* and *resource management*. The solution proposed in the paper focuses on container level memory management. To achieve the same, the authors propose two algorithms: *RelM* and *GBO*. *RelM* is an algorithm which improves on the workings of garbage collectors in containers. *GBO* on the other hand, uses the analytics from *RelM* and optimises the pre-existing Bayesian Optimization algorithm. This improvised algorithm is Guided Bayesian Optimization (*GBO*) algorithm.

- **iBTune: Individualized buffer tuning for large-scale cloud databases.** [TZL+19]: The *iBTune* algorithm proposed in the paper focuses on cloud databases. This comes with the difficulty of having different buffer size requirements for different DB instances. To overcome this and provide optimal results, The algorithm focuses on the relationship between the miss ratio and buffer pool size. In addition to that, the algorithm implements a Deep Neural Network (DNN) algorithm to predict the upper bounds of response times. To aid the workings of the DNN algorithm, *iBTune* collects the metrics at an interval of a second and analyses the time series data. Once the tuning is done, the changes are implemented online without affecting the running instances and programs. This also is rolled back when the performance drops compared previous settings.

- **A new approach to dynamic self-tuning of database buffers** [THTT08]: This paper proposes an analytical method of using ”Buffer Miss Equation” to automatically tune the database buffers. Using this equation, the authors aim to improve gradient descent method, rapidly partition the buffer, reduce the number of page misses. The paper further goes on to show case the validity of the equation and the equation’s robustness to integrate with
multiple existing buffer replacement algorithms. Using the equation, the author is also able to successfully optimise the gradient descent algorithm.

- **Goal-oriented buffer management** [BCL96]: The paper proposes a revised version of the buffer management algorithm ”Fragment Fencing” called ”Class Fencing”. Based on the assumption that the reduction of miss rate would improve the response time, Class Fencing algorithm uses hit rate concavity to predict the hit rate. The hit rate concavity is an implementation of concavity theorem. According to concavity theorem, optimal buffer replacement policy would not increase the hit rate with the addition or increase of memory to the buffer. Further, using the concept, the algorithm predicts the future requests using two previous observations in order to improve the accuracy and reduce miss rate.

### 2.2.3 Field Study:

This section introduces a paper which made a field study of AI based configuration tuning solutions. This paper is on the similar lines of this thesis and yet has with a very different goal. And hence, this section is dedicated towards the highlighting the paper.

- **An Inquiry into Machine Learning-based Automatic Configuration Tuning Services on Real-World Database Management Systems** [VAYB+]: This paper presents a field study which evaluates the performances of the groups of AI models against manual configuration tuning. The paper brings light to the improvement seen with the configuration tuning with the help of AI. The paper specifies that workload and system complexities and operating environments hold back the AI configuration tuning to perform as efficiently in real world cases as in test scenarios. The paper, similar to this thesis, uses [VAPGZ17] as the main architecture. Implements and compares Gaussian Process Regression, Deep Neural Network and DDPG algorithms on the system of [VAPGZ17]. Unlike this thesis, the paper focuses on the field study and emphasizing on the importance of the AI in configuration tuning.
Also, unlike the paper, this thesis focuses on expanding scope in terms of workloads, agents and optimization goals. This thesis also aims to contribute in some of the unexplored areas of the research.

Summarising all the above mentioned papers and its brief overviews, it is evident that the database configuration and memory allocation tuning posed as a problem for manual tuning from decades. This had given rise to automation solutions. During recent years, to optimise the automated solutions, researchers are headed towards the usage of Artificial Intelligence (AI).

Among the AI models, the most commonly used ones are: Regression, Bayesian Optimization, Deep Deterministic Policy Gradient (DDPG) and Deep Neural Networks (DNN). Among these classes of AI models, the one which proved most effective was DDPG which is a Reinforcement Learning (RL) algorithm. It should be noted that DDPG is just one of the RL algorithms and many other algorithms in RL class of AI. There exists and have been discovered many other RL algorithms which posses similar characteristics and perform better.

Having said that, it is also worth noticing that all the solutions focused on then existing workload patterns. Even though the variation in the workloads were noted, least to none focus was drawn on hybrid workload patterns.

Another important aspect that has been not explored in the above papers is the optimization of multiple objectives. As per my knowledge, research till date, has focused on improving a single objective or a metric (throughput or latency or hit rate or accuracy). The algorithms fail to improvise more than one objective simultaneously.

These points constitutes the major areas which are left unexplored today. This thesis tries to pave path of exploration in one or more of these areas. Further on the open topics tackled by the this thesis is explained in the section Section 2.5

2.3 AI models used for configuration tuning

This section introduces the topics and explains the concept of reinforcement learning. Reinforcement learning is a broad area of AI which
has been explored to be applied for database requirements. Among the vast algorithms within reinforcement learning, this section provides a brief explanation of the algorithms that are relevant to this thesis.

2.3.1 Reinforcement Learning

Reinforcement Learning (RL) is an AI technique. The goal of this technique is to get the optimal result for all its actions by having a balance between exploration and exploitation. RL iteratively learns this how to get the optimal final result based on small steps. These steps can be called actions and each iteration is called an episode. There are main components of the RL algorithm. These components are:

- **Environment**: Environment is where the agent exists [Mit97]. In other words, everything outside of agent that agent interacts with is called the environment for the given RL problem [SB18].

- **State**: The representation of the agent in terms of its properties at any given time step $t$ is the state of the agent.

- **Action**: Action represents the task performed by the agent in order to move to its next selected state.

- **Reward**: Reward is the effect that agent experiences for choosing the action and the state. It can be positive or a negative reward usually depending on how close of farther the agent is from finding the optimal state-action path towards the goal.

- **Agent**: Agent represents the entity that is learning the policy to find the optimal way to reach the goal in the environment. For every action of the agent, a reward is generated within the environment.

Figure 2.10, depicts the cycle of the reinforcement learning algorithms. The figure captures all the tangible components of the RL algorithm. Through this iterative process, the agent learns the policy based on which, agent chooses which action to take in a given state.

Policy represents the relationship between the state-action and reward functions as understood by the agent. This policy gets updated iteratively until the agent learns the optimal policy. The optimal policy
2.3. AI models used for configuration tuning

Figure 2.10: Cycle of reinforcement learning [RLC]

represents the best actions that needs to be taken in any given state of the agent. Choosing these best action of a given state will further lead the agent to get the optimal cumulative reward. This forms the core of the RL algorithms.

One of the most primitive implementation of RL is Q-Learning algorithm. Q-Learning algorithm starts by initializing all the state-action pair rewards to 0. For the first iteration, as all the reward values are 0, the action-state chosen will most likely be random. In other words, the agent explores the environment. As and when the agent reaches a particular state, the reward of that corresponding state is updated. Once the agent has the reward values known, the agent exploits the environment to find the optimal policy. To find the optimal policy, for each state, agent chooses the next state by calculating the expected optimal cumulative reward of every possible state-action pair from the given state. Once the agent has the optimal policy for each state to reach the final goal, the agent stops learning.

In the Q-Learning (RL in general), it can be seen that the algorithm takes discrete value for states and actions. Most of the real world scenarios are not always discrete valued states. In order to reflect the actual problems, the algorithms should be able to work with continuous valued domain. To overcome this drawback, RL was modified to support continuous valued action space.

It should also be noted that the Q-Learning algorithm and most of the RL algorithms in general, focus on smaller set of states and actions. Most of the inter-disciplinary applications of RL cannot be appropriated with this drawback. Hence to overcome this and expand
the applications of RL, RL needs to be modified to support higher action-state space. In order to do that, Deep Learning (Deep Neural Network) is used in RL. Deep Learning enables RL to support higher state-action pair and provides better search of optimal results. This method of using DL in RL for finding the optimal policy is called Deep Reinforcement Learning.

Deep Reinforcement Learning is being explored to be applied in Database in order to automate most of the DBA’s work. Most of the works proposed in [LZLG19], [ZLZ+19] are the algorithms which help in configuring the database with the help of Deep Reinforcement Learning (DRL) algorithms.

Figure 2.11: Classification of Deep Reinforcement Learning Algorithms [LCP+21]

2.3.2 Deep Deterministic Policy Gradient (DDPG)

Figure 2.11 shows the characteristics upon which the DRL algorithms are classified. Among the DRL algorithms, the most commonly used in database configuration tuning as of today is DDPG (Deep Deterministic Policy Gradient) [LHP+19] algorithm. DDPG is a Model-free, off-policy and policy-gradient based algorithm.

For a very long time, Reinforcement Learning algorithms used discrete action space for workings. This constricted the applications of RL in other diverse fields. In order to overcome this constraint, RL algorithms were adapted to be used even in continuous action spaces.
As we know, the database configuration knobs are mainly continuous valued. With the discovery of Deterministic Policy Gradient (DPG) \cite{SLH+14} algorithm, the applications of RL in database configuration tuning was possible.

Deterministic Policy Gradient (DPG) \cite{SLH+14} algorithm forms the basis of DDPG algorithm. The main aim of the DPG algorithms is to find the minima/maxima based on the policy gradients in the action space. The same is true for DDPG as well. DDPG uses actor-critic approach to solve the problem similar to DPG \cite{SLH+14} algorithm. The major difference in DDPG is that the Deep Neural Networks are used to learn the state and action spaces and hence approximate the policy gradients.

### 2.3.3 Maximum a Posteriori Policy Optimisation (MPO)

MPO \cite{AST+18} is a Deep Reinforcement Learning algorithm which focuses on optimising the target objective by supporting continuous values for actions and states. The algorithm also supports comparatively larger number of states which enable us to adopt to database tuning problems. MPO transforms the given RL problem into an inference problem. Inference problem helps us have the minimum assumption of the trajectory distribution of parameters. With this minimum assumption about the parameters, the algorithm is able to perform more efficiently.

The MPO algorithm is an off-policy algorithm. Off-policy algorithms comes with two different policy networks to learn the the actual policy - target policy network and behaviour policy network. Having two networks to learn the policy, enables the agent to make a better trade-off between the exploration and exploitation during the search for optimal action. Behaviour policy is used to explore the environment and guide the target policy towards the global minima/maxima.

The MPO algorithm trains and learns the policy in two main steps: E and M steps. In E step, the algorithm focuses on re-weighting the action-state samples. Action-weight samples helps infer the trajectory distribution of parameters. Hence, choosing the relevant and better rewards samples help converge sooner. In M step, the algorithm focuses on updating the parametric policy using the deep neural network based on the updated action-state samples.
In other words, the algorithm works by assuming that the optimal solution is obtained. With the assumption of the optimal solution, algorithm calculates the probability of the actions in the current state. The actions with the maximum probability is chosen. The goal of this algorithm is to learn the policy which provides higher probability to the better actions.

2.4 Transactions vs Analytics

With the widespread use of databases, the databases have transformed to adapt different requirements. These requirements span from the types of queries, amount and speed in which it needs to be executed. Based on these criteria, the current distribution of workloads handled by databases are as follows:

- TPC-C: Benchmarking for On-Line Transaction Processing Section 2.4.1f
- TPC-H: Benchmarking for On-Line Analytical Processing Section 2.4.2

Each of these workloads have different properties and serve different purposes and hence are used in different business applications. The following subsections, draw a details view of the workloads.

2.4.1 TPC-C

On-Line Transaction Processing workloads are focused on the business requirements which require fine recording. In other words, the businesses which needs tracking of the processes that happen typically produce OLTP workloads. OLTP, as the name suggests, focuses on transactional queries. The transactional queries are mainly insert, delete and update statement which are usually pointed to single records at a time.

Some of the examples of the OLTP workloads include - bank transactions, online purchase transactions, booking travel tickets. etc. If observed, all these business system requirements have one thing in common. These systems are mostly catered to a group less than 10 people (records in the databases) at a time. On the other hand, it should also cater to more than a million groups of users at the time.
In other words, OLTP workloads should cater to large number of user simultaneously access less than 10 number of records. OLTP system users either create an account or book a ticket, delete or modify the existing account or tickets. These requests are transformed into insert, delete and update queries on the databases. Hence falling into the OLTP workload category. OLTP usually uses relational databases to enhance the performance for the workloads.

As established, the OLTP workloads are very frequent transactions with minimal number of records for each transactions. This property of the workload gives rise to the requirement of the DB to be compliant to maintain concurrency and atomicity. Concurrency ensures that more than 1 user can use the database without affecting the transactions of the other user. Atomicity makes sure that the transactions are either complete or not started at all. This in turn makes sure that during the transaction, either all the queries in the transaction are executed or none are executed.

With more and more applications using OLTP workloads and databases, satisfying the basic need of atomicity and concurrency does not suffice. To be on top of the game, the performance of the system is measured. As the OLTP workloads’ main focus remain the frequent small sized transactions, the speed in which is it executed is measure. The most suited metric to answer this question is throughput. Throughput is a metric which gives us the measure of transactions in a unit time period. Throughput gives the reader the insight of the speed of the database. More specifically, the response time of the database for a given query or a transaction.

Even though the unit of measurement mostly is throughput for OLTP workloads, there needs to be standardization. To have a standard way to measure the OLTP workloads, OLTP benchmarks are created. The paper [DPCCM13] provides an optimal overview of the benchmarks for OLTP. The benchmark used in this thesis to evaluate the performance OLTP is TPC-C. As mentioned in [TPC16a], TPC-C uses 5 concurrent transactions of varying types and complexity. Using 9 tables in the database with varied range of record and population sizes, TPC-C uses tpmC (transactions per minute) to evaluate the performance.

The benchmark used to evaluate the performance of the workloads has a set of configurations. These configurations are to be modified
to reflect the workload characteristics against which the DBMS performance is evaluated. The table Table 2.2 provides the list and the description of the parameters for the OLTP benchmark.

<table>
<thead>
<tr>
<th>Workload Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scale Factor</td>
<td>Determines the number of warehouses. The number of warehouses is directly proportional to the database size and in turn influences the throughput. [TPC16a]</td>
</tr>
<tr>
<td>Terminal</td>
<td>Determines the number of terminals supported during workload execution. Terminals are components which enable the user to input commands or view the transaction outputs. [TPC16a]</td>
</tr>
<tr>
<td>Time</td>
<td>Specifies the duration of workload execution</td>
</tr>
<tr>
<td>Rate</td>
<td>Indicates the maximum throughput the system can achieve. Rate of transactional queries made to the database.</td>
</tr>
<tr>
<td>Weights</td>
<td>Defines the distribution of type of transactions. The weights specify the minimum requirement for each transaction type from each terminal.</td>
</tr>
</tbody>
</table>

Table 2.2: TPC-C workload benchmark parameters [TPC16a]

2.4.2 TPC-H

Online Analytical Processing workloads forms the set of queries which are executed to analyse the data. Businesses which use the data collected during the transactions to analyse and draw business insights, focus on OLAP workloads. Figure 2.12 helps understand the differences between OLAP and OLTP. This also provides an insight about the properties of OLAP workloads.
Analysis of the data is to be done on cumulative values drawn from the transactions. To derive these cumulative values, the queries executed are mostly aggregated on large number of records. As a result, the OLAP workloads consists of queries which affect large number of records. With analysis either takes place once per day or a week, the frequency is much lesser than OLTP transactions.

OLAP transactions manipulate large number of records as part of every request. Hence, the databases supporting OLAP transactions focuses on executing a single query with minimum duration. This infers that, *elapsed time* the time taken to execute the large sized queries is the criteria to evaluate OLAP transactions. To enable this, the databases to support OLAP are mainly columnar databases. Columnar databases help retrieve the filtered records faster compared to the row based database architecture.

Similar to OLTP, to measure the OLAP transactions, there exists a benchmark. The paper [CFG+11] provides a guideline of the benchmarks used for OLAP. In this thesis, TPC-H benchmark is used to evaluate the OLAP workload transactions. Table 2.3 provides the list and description of the configurations for the benchmark.

<table>
<thead>
<tr>
<th>Workload Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scale Factor</td>
<td>Specifies the size of the database. The records in the table are proportional to this value. [TPC16b]</td>
</tr>
<tr>
<td>Terminal</td>
<td>The number of terminals the database needs to support during the workload execution. Terminals constitutes the user interfaces with the ability for input and output [TPC16b]</td>
</tr>
<tr>
<td>Time</td>
<td>The duration for which the workload is tested on the database.</td>
</tr>
<tr>
<td>Serial</td>
<td>There exists streams to test the throughput. The stream values are fixed proportional to the scale factor. This component specifies if the queries from different streams needs to be executed serially or parallelly.</td>
</tr>
<tr>
<td>Rate</td>
<td>Specifies the upper-bound for the throughput of the database under test.</td>
</tr>
<tr>
<td>Weights</td>
<td>Indicates the workload the distribution of the transaction types. [TPC16b]</td>
</tr>
</tbody>
</table>

Table 2.3: TPC-H workload benchmark parameters [TPC16b]
2.5 Open directions in AI for configuration tuning

Configuration tuning in DB is one of the most important topic due to its effect on performance and difficulty to tackle. It is also a vast topic that has high scope for innovation and new ideas. With the collaboration of AI to help achieve optimal configuration tuning, it opens hundreds of doors for researchers. This section discusses few such areas within AI for configuration tuning that are least explored or unexplored.

With the knowledge of current research trends in AI and DB, and the awareness of need for automation in DB has pushed us to explore certain areas for improvement. These topics have unfortunately been least explored in the research field currently. In order to bring light on the topics and help improve the work within, few open directions in AI for configuration tuning are as follows.

- **Hybrid workload tuning:**
  There exists two main workloads: OLAP and OLTP which focuses on different needs of the system and data. For each of the workload, different characteristics of the DB is preferred to aid optimal performance. With the advent of high number of small scale companies and businesses, it can be challenging to have different databases for different workload types. Hence, a hybrid workload which uses both OLAP and OLTP transactions were introduced. Further details about the workloads and the materials used is explained in detail in section ??

On configuration tuning front, the tuning algorithms and systems currently focuses on the main workload types - OLAP and OLTP. As HTAP workload type is relatively new, least to none contributions are made in configuration tuning for HTAP workloads. This thesis also focuses on tuning HTAP workloads and hence making way for more contributions in the direction.

- **Multi-objective optimization tuning:**
  Existing automated configuration tuning systems works towards optimization of one metric or a feature of the database system. In reality, that is hardly the case. The users of the DBS would
require the system to perform best in terms of more than one metric.

The DBAs today spend hours and days optimising more than one metric. In order to actually help DBAs, the configuration tuning systems should be capable of tuning in order to optimise more than one metric. Hence rises the need for multi-objective optimization tuning.

This thesis attempts to progress in the direction and introduces the multi-objective optimization in the form of agents. The source of study to implement the multi-objective optimization agents is [AHH+20]

- **Time series:** Time series constitutes a sequence data which are indexed at a given time interval. Time series data are extensively used today to draw useful information and monitor the variation of activities in companies and businesses.

Time series is relatively new form of data which can be successfully merged with database technologies.

- **Database for Time Series data:** As we all know that databases provide immense support to organise and manipulate traditional unstructured data. But the same is not true for time series data. The support for time series data is yet to be easily accessible, optimal and universal. Fortunately, with the trend of high usage of time series data, databases are bringing the solutions which help provide optimal support to time series data usage. [Bad] provides an overview and the comparison of the databases which currently provide support for time series data. These solutions are comparatively recent and yet to meet all the requirements optimally.

- **Time Series optimization for Database:** Time series data is currently being used by most of the applications to optimise the performance. Time series data provides and insight of the pattern and hence helps deduce more information to optimise applications. The success of the usage of time series data in optimization can also be implemented with databases.
Today, in order to optimise the performance of databases, varied automation techniques are being implemented. The best performing solution itself takes hours to produce a working solution. This has given rise to optimization of the same solutions. Time series data can help database researchers solve this solution.

Optimization of database involves many facets like: SQL query tuning, database configuration tuning, workload tuning and scheduling, data distribution etc. [Bur] proposes the tuning of SQL queries using time series data. [TZL+19] uses the time series data to tune the buffer sizes in a large scale cloud databases. This is just a couple of facet and a use cases in database performance tuning area. The same can be extended and implemented to all configurations tuning of the databases and other facets as well.

2.6 Summary:

Every database comes with hundreds of configuration tuning knobs. These knobs vary from database to database. The value ranges of these knobs differ for different hardware settings. In order for the database to work efficiently for a given workload, the configuration knobs needs to be set to the right values. This task, traditionally done by a DBA, is very error prone and overwhelming for a human. Hence there are solutions to automate the configuration tuning ([LZLG19], [VAPGZ17], [DTB09], [ZLG+17], [KB20], [ZLZ+19], [YG]).

The solutions are moving towards RL algorithms, mainly DDPG. This leads to other RL algorithms not being explored. The existing solutions focus on the two major workload types: OLTP and OLAP. The tuning for hybrid workload is not yet supported. Also, these solutions focus on tuning and optimising a single metric. Even though there are instances of scalarization techniques being used, multiple objectives itself are not being optimised.
3. Architectural Overview and Research Questions

This chapter provides an overview of the research questions addressed in this thesis. It also explains the underlying architecture of the system.

The structure of the chapter is as follows:

- **Research Questions:** Section 3.1 describes the research questions that form the basis of this thesis.
- **System Architecture:** Section 3.2 provides an overview of the system on which research questions are executed and tested.
- **Workloads and Metrics:** Section 3.3 explains the context of the research questions in terms of workloads and metrics.

### 3.1 Research Questions

This section presents the focus and purpose of this thesis. Below listed are the research questions which will be answered through the course of this thesis.

- **1. Efficiency of Exploration**
  To what extent do different methods for exploration, including reinforcement learning, neural networks and contextual multi-arm bandits contribute to discovering the optimal configuration given
a limited number of time-steps? Do one method work better for all workload types? Do the number of configuration knobs tuned have an influence on the performance?

- **2. Influence of start population on convergence**
  Do start population techniques have a significant influence on the optimal solution obtained in fixed time-steps over all the workloads and agents? To what extent does the start population techniques affect the performance of the tuning algorithms? Does the implementation of more exhaustive start populations have an influence in the performance of exploratory agents?

- **3. Best of Scalarization**
  Do each of the scalarization techniques have different influence on the optimal solution obtained in fixed time-steps? Is there a scalarization technique that works better for all the considered DBMS? Which scalarization technique is best suited for the problem being addressed in this work?

- **4. Multi-Objective Optimization**
  What is the influence of optimizing multiple objectives simultaneously as opposed to scalarised optimization of the objectives in fixed time-steps? Is multi-objective optimization technique better than scalarization techniques?

### 3.2 System Architecture

This chapter provides a detailed information about the underlying system. In order to provide a clearer view of the system used for the thesis, this section is divided into explanations focused on:

- **Architecture of Otto-Tuner:**
  OtterTune [VAPGZ17] is the underlying system on which the work of this thesis is built on. In order to explain the changes of this thesis better, Section 3.2.1 provides a brief explanation of the otterTune system.

- **Transformation of OtterTune to Otto-Tuner:**
  To answer the research questions of this thesis, the otterTune system architecture had to be modified. These modifications are explained in Section 3.2.2.
3.2.1 Architecture of Otto-Tuner

Otto-Tuner is a recommender system which run on the client database metrics to analyse and recommend an optimal database setting. Otto-Tuner is an extension of otterTune [VAPGZ17]. The work in this thesis (Otto-Tuner) uses the architecture and the system of OtterTune [VAPGZ17]. As explained and designed in [VAPGZ17], OtterTune helps the DBA to arrive at the best possible database settings for a given target objective.

![OtterTune Architecture](image)

As can be seen in Figure 3.1, the system architecture consists of two main components - Server and Client. Server handles the data collection, processing, analysis, and configuration recommendation. Whereas the client takes care of the data collection and implementation of the data needed and provided respectively, by the server. The interaction between these two major components is made possible with the help of HTML-POST and HTML-GET messages.

Data is stored and processed in the server. This component comprises of tuning agents. Tuning agents work on the analysis of the processed data. Based on the analysis, the next configuration is suggested to the client. Server also provides a web Interface for the user to interact with the system. This interface ensures the database administrator can use the system optimally with minimal knowledge of internal workings of the system.

Client being the other major component of the system, works with the database which needs configuration recommendation. Client acts as the representative of the target database to be tuned. The data needed
by the server in order to recommend a setting, is collected at the client side of the system and communicated to the server. Client database is the target database which needs tuning for the better performance of the database.

Client driver is the key point of contact between the client and the server. This component integrates the system by acting as the sheer mediator between the client-database, client-controller and the server.

3.2.2 Transformation of OtterTune to Otto-Tuner

This section brings light on the changes made to the underlying system in order to expand the workings of the otterTune. The five major changes made on the otterTune as part of the thesis, helps as the backbone in answering the research questions. Figure 3.2 shows these five components. This figure is drawn similar to Figure 3.1 which describes the base architecture of ottertune. This similarity is maintained to help reader gain a visual understanding of the changes made to the system. The changes made for the components 1,2,3,4, and 5 are further explained in detail in the sections Section 3.3, Section 3.2.2.1 and Section 3.2.2.2, Section 3.2.2.4, Section 3.2.2.5, and Section 3.2.2.3 respectively.

Figure 3.2: Otto-Tuner (Extended OtterTune) Architecture: Depicting the five major changes made to the original ottertune [VAPGZ17] architecture.

3.2.2.1 Support higher versions of client database

OtterTune maintains the system supported database types and its version details, stored in the server database - MySQL. As it is known, every database and every version of the same database comes with
different set of metrics and configuration knobs. For every version of each of the database type, the otterTune maintains the supported and required metrics and configuration knobs. The DBA is also provided with an option to choose among the supported versions of databases for tuning. To ensure the metrics and knobs match the database type being tuned, otterTune also makes a check to verify the version and database type provided by the database metrics against the configured values in the session. This test helps otterTune to be less prone to errors during the tuning. It also ensures the tuning provides the DBA, the best configuration possible. The changes in this section is depicted by the component 2 in the extended architecture Figure 3.2.

As the system works closely with the database types and versions, enabling the system to support a new version of the database requires changes in all the mentioned aspects. Below are the specific steps taken to add support to multiple version of PostgreSQL database during the course of the thesis.

- **Add the new version in the catalog model:** The otterTune maintains all the properties of a session in terms of models. During the creation of the session, the DBA selects a value for each of the session’s properties. Database type and version is an important property of the session. Adding the new version in this model, enables the version to be displayed in the web interface for the DBA.

- **Create knobs file website/fixtures:** It is necessary to update the configuration knob changes for the given version. In order to do so, a json file is created. This file contains all the available and required configuration knobs from the version of the database. For each of the configuration knobs, information about the knob attributes has to be provided. Knob attributes are listed in Table 3.1. Figure 3.3 is the sample configuration knob information snippet.

- **Create metrics file website/fixtures:** Create a file which represents the configuration metrics in the website/fixtures. For each of the metric, details about its attributes has to be provided. The metric attributes are listed in Table 3.2. Figure 3.4 is the example metric information snippet.
Figure 3.3: Knob Attributes Example

```json
{
    "fields": {
        "category": "Client Connection Defaults / Statement Behavior",
        "maxval": "2000000000",
        "dbms": 8,
        "name": "global.vacuum_freeze_table_age",
        "minval": "0",
        "default": "150000000",
        "tunable": false,
        "enumvals": null,
        "vartype": "2",
        "context": "user",
        "scope": "global",
        "summary": "Age at which VACUUM should scan whole table to freeze tuples.",
        "unit": "3",
        "description": "",
        "resource": "4"
    },
    "model": "website.KnobCatalog"
}
```

Figure 3.4: Metric Attributes Example

```json
{
    "fields": {
        "dbms": 8,
        "default": 0,
        "scope": "global",
        "vartype": 2,
        "name": "unified_HTAP_metric.QphH",
        "summary": "tpcc metric",
        "metric_type": 1
    },
    "model": "website.MetricCatalog"
}
```
3.2. System Architecture

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>The name of the configuration knob</td>
</tr>
<tr>
<td>Category</td>
<td>Specifies in which category this knob gets used during the execution of the workload on the database. Ex: Logging, Reporting, Replication,</td>
</tr>
<tr>
<td></td>
<td>collections, connections, query tuning etc.</td>
</tr>
<tr>
<td>Maximum Value</td>
<td>This specifies the upper bound of the knob value for the given DBMS</td>
</tr>
<tr>
<td>Minimum Value</td>
<td>This specifies the lower bound of the knob value for the given DBMS</td>
</tr>
<tr>
<td>Summary</td>
<td>This value summarises the effect of the knob on the DBMS.</td>
</tr>
<tr>
<td>Scope</td>
<td>Some of the knobs can different values per session making its scope to be session wise or local or global. This value specifies the scope of the knob.</td>
</tr>
<tr>
<td>DBMS</td>
<td>OtterTune maintains and supports tuning of multiple databases. This variable in the knob configuration detail, specifies which database this</td>
</tr>
<tr>
<td>Tunable</td>
<td>configuration knob belongs to.</td>
</tr>
<tr>
<td>Context</td>
<td>Specifies if this knob comes under the context of user or sighup or superuser or postmaster.</td>
</tr>
<tr>
<td>Description</td>
<td>Describes the importance, usage and effect of the knob for the given DBMS.</td>
</tr>
<tr>
<td>Resource</td>
<td>As databases have plenty of knobs, it can be grouped into memory, CPU, storage, threads etc. This value denotes the resource the knob affects.</td>
</tr>
<tr>
<td>Unit</td>
<td>This indicates the unit of the configuration knob value.</td>
</tr>
<tr>
<td>varType</td>
<td>This feature indicates the value type of the configuration knob. It can be either numerical (integer or float), categorical (ENUM), boolean or</td>
</tr>
<tr>
<td></td>
<td>string.</td>
</tr>
</tbody>
</table>

Table 3.1: Knob Attributes as stored in Otto-Tuner

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>The name of the performance metric.</td>
</tr>
<tr>
<td>Metric Type</td>
<td>This field specifies if the metric is of counter type or maintains statistics or just provides information to the reader.</td>
</tr>
<tr>
<td>Datatype</td>
<td>Specifies the value type of the metric. The possible values being: Boolean, integer, string, real number, ENUM or a categorical value.</td>
</tr>
<tr>
<td>Summary</td>
<td>This field describes the metric. Also, Indicates the meaning and importance of the metric within the DBMS.</td>
</tr>
<tr>
<td>Scope</td>
<td>Specifies if the metric belongs to database or local or global scope.</td>
</tr>
<tr>
<td>Default</td>
<td>This field specifies the default value the metric takes.</td>
</tr>
<tr>
<td>DBMS</td>
<td>Specifies which database this metric belongs to. As ottertune maintains and is capable of tuning more than 1 DBMS, this value helps maintain</td>
</tr>
<tr>
<td></td>
<td>data from different DBMSes.</td>
</tr>
</tbody>
</table>

Table 3.2: Metric Attributes as stored in Otto-Tuner

- **Migrate the changes to MySQL:** OtterTune maintains MySQL as the main database of the system. In MySQL, the data related
to supported databases, versions, metrics and knobs are stored. During execution of the system, these values are fetched to verify the support of the database. Hence the added json files of metrics and knobs and the catalog model data needs to be updated in MySQL. This is done at the server side with the help of \textit{django} and \textit{python}.

\subsection*{3.2.2.2 Support for memSQL database}

As part of the expansion made to the otterTune, memSQL database support is added to the otterTune architecture. The changes required to add another database is quite similar to adding a new version of a database. Hence, changes in this section is depicted by the component 2 in the extended architecture Figure 3.2. Though it is similar, it is not all that needs to be done. Apart from the 4 steps mentioned in the section Section 3.2.2.1, the below step is performed.

- **Knobs and metrics:** The number of times mentioned, does not match to the burden of the fact that - every database has different set of knobs and configurations. Adding a new database, requires to know the differences between the two databases in terms of knobs and configurations. It is also necessary to find which metrics are supported by the database. Once we have the list of metrics, it is important to note the unit of the metrics. These information has to be configured into the system to help system understand the database performance.

With the understanding of the database performance, arises the question, \textit{Which knobs to tuned?}. Not all knobs are tunable by all the databases and not all can be tuned with the restrictions of the system hardware. It is required to know the tunable knobs. With this information, we can ask the next question, \textit{What are the tunable values for these knobs?}. Similar to the tunable knobs, the value ranges for these knobs differ from databases to databases. It is also heavily dependant on the hardware on which the database is running.

If the configurations are not within the tunable range of knobs and values, the database crashes. In order to avoid this, it is required to find out the knobs and values and configure ottertune
accordingly. This helps avoid the mishaps of database crashes during the tuning session of the database.

### 3.2.2.3 Sampling techniques

Sampling techniques define how a point or a configuration is obtained and recommended from a search space by the ottertune server. This section provides us an overview of where sampling technique is used and where the changes were done to add different sampling techniques. How the changes in this section affect the ottertune is depicted by the component \(5\) in the extended architecture Figure 3.2

The use of sampling techniques applies only to the specific tuning agents. Figure 3.5 provides an overview of the information flow for the tuning agents - Neural Networks and Gaussian Process Bandits. As can be seen in Figure 3.5, the HTML POST from the client triggers the process on the server side. This POST message contains the results obtained after the execution of an iteration of target workload benchmark queries. After validating the results obtained from the client, server starts the process of tuning the model and recommending the next configuration to the client.

The model tuning process depends on the tuning agent (Exploratory tuning algorithm) chosen by the DBA while creating the session. As part of the tuning process, the server checks which tuning agent is mapped to the session being tuned. Based on the selected tuning agent, the corresponding workflow is chosen. For the tuning agents - Neural Networks and Gaussian Process Bandits, the steps leading to the respective algorithms are the same. Hence, the flow chart is generic for both the tuning agents.

In Figure 3.5, the blocks marked in green represents the blocks that were modified as part of the thesis. The process block for sampling technique is of interest in this section.

Depending on whether the model trained for the session is stored or not, the model parameters are updated or created. Once the model is trained, the sampling techniques are used. The sampling techniques provides the points in the solution space created by the configuration knob boundary values. These sampled points in turn are transformed into the next recommended configurations for the client, depending
on the point’s error value prediction. These points’ performance are predicted using the session’s trained tuning agent.

Web interface is used to help the DBA to choose hyper-parameters that are to be used for the given tuning session. To enable the DBA to choose which sampling technique to use, the type of sampling technique has to be given as a hyper-parameter using the web interface. The sampling technique is not used by all the tuning agents ottertune supports. This is the reason, it is used as an hyper-parameter for the tuning agents to which it applies.

3.2.2.4 Multi objective scalarization techniques

OtterTune is an automated database configuration tuning system which can be relatively easily extended to support more features. One such feature is tuning the configurations with multiple objectives. This section provides the steps that were taken to add this feature with the help of scalarization. Where does the changes in this section fit in the ottertune architecture is shown by the component 3 in the extended architecture Figure 3.2.

The steps taken to add multi objective scalarization are:

- **Web Interface:** As mentioned in the earlier section, ottertune provides a web interface which helps the DBA pick the configuration he/she intends to have for the tuning. Any changes made should be made accessible to the DBA through the web interface. In order to do so, new metric is added and this is made available in the web interface. Further, the session with this linked metric is directed to the newly added metric function to perform scalarization. It is also possible to have different string as names for the target objective to be displayed in the web interface. This helps to have an isolation between the back-end functionalities. It also spares the DBA and enables to have a better understandable name for the target objective.

- **Addition of metric function:** Ottertune in terms of software architecture, treats the target objective as a class. This class is mapped to the database in which it can be used. Any addition of new target objective should be declared as a function. Each function takes the hyper parameters and the metrics as the function
Figure 3.5: Information Flow in OtterTune
parameters. This enables to have customised target objectives. The metric values obtained as a parameter can be manipulated to arrive at the target objective value. These manipulations, class functions, and database mappings were added as part of this step to have new metrics in ottertune. The most important information about the metric, whether the metric value "lesser is better" or "more is better" is also declared as part of the target objective class.

The block of code this section represents in marked in green colour in Figure 3.5. Once the data is validated by the server, the server goes on to calculate the session’s target objective value. During this phase, the changes made in this section will be executed.

3.2.2.5 Multi objective MPO agent

The previous Section 3.2.2.4 explained how multiple objectives were enabled to be handled in ottertune. On the similar lines, this section explains another method to handle multiple objectives in ottertune. Apart from the technique which help convert the multiple objective algorithms to single objectives, there exists algorithms which are designed to handle multiple objectives. This section explains the changes made in order to accommodate one such method in ottertune. The component 4 in the extended architecture Figure 3.2 shows how this addition fits in the ottertune architecture.

The major changes that were made to support another tuning agent are:

- **List of algorithms:** Ottertune maintains a list of algorithms supported in the system. This list is displayed in the web interface for the DBA to choose from. If the logic is added in the back-end and not added in this list, the DBA will not be able to choose the tuning agent and hence will not be of much use. To ensure a complete working, the new tuning agent has to be added to the list of algorithms.

- **Redirection:** As mentioned in Figure 3.5, after validating and pre-processing the received metrics, the server chooses the next workflow. The workflow chosen is completely dependant on the
3.2. System Architecture

Figure 3.6: Information Flow in OtterTune for MPO agent
tuning agent selected by the DBA during session creation. For the program to choose the added tuning agent, the corresponding decision block has to be altered. The added option should point to the workflow required for the new tuning agent. This refers to the diamond block marked in green colour in the flowchart Figure 3.5.

- **Execution:** Once the program is redirected to the proper workflow, there should exist an error free processing steps to have smooth workings of the system. These steps range from transforming the metrics as accepted by the tuning agent to training the agent and querying the next recommendation.

Figure 3.6 provides the reference to understand the blocks of code that were changed. This flow chart is a continuation of Figure 3.5. The symbol 'A' represented in a circle of green symbolises the continuation from the previous workflow. In both the figures, the blocks marked in green represents the code blocks that were changed as part of this thesis.

The blocks marked in green in the flow chart Figure 3.6 represent the steps as explained in the point Section 3.2.2.5. In order to train the agent, apart from regular normalization of data, the data also needs to added in the replay table. Replay table is the table which acts as the reference to the tuning agent. Using this reference, the tuning agent draws conclusions about the relation between the actions(configuration knobs) and the rewards (metrics). This process of finding the relation requires an environment and 'n' iterations of processing. The same has to enabled and handled in the code before being able to use the tuning agent. The tuning agent then iterates for 'n' steps and recommends the next action. This action is converted to the configuration metrics and recommended to the client server to be used on the database.

### 3.3 Workloads and Metrics

So far, this chapter covered the changes to the otterTune which were directly related to its architecture. This section draws attention to the workloads and metrics which were added to the otterTune. Hybrid Transaction and Analytical Processing(HTAP) represents the modern databases which are capable of performing both OLTP and OLAP.
transactions on the same database [GS18]. These databases are being extensively used today due to its diversity. Unfortunately, this workload type was not tunable readily in the ottertune. This thesis has included the benchmark tool which helps evaluate and tune hybrid workloads. The component 1 in the extended architecture Figure 3.2 shows how this addition fits in the ottertune architecture.

In order to fit this hybrid benchmark, HTAPBench [CPV+17] was installed on the system, HTAP supportive database was created in the target DBMS and the configuration changes were made to invoke the benchmarking tool. HTAP being the hybrid workload type, the metric used to measure the performance of HTAP should represent both OLTP and OLAP aspects. To be able to measure all aspects of the database, a new metric was designed by [Paw19].

\[
QpHpW = \frac{QphH}{OLAPworkers@tpmC} \tag{3.1}
\]

Equation 3.1 was formulated by [Paw19]. In this metric, the value $QpHpW$ stands for *Queries of type H per Hour per Worker*. In other words, the metric represents the number of queries executed by each of the OLAP workers at the registered $tpmC$. As can be seen in Equation 3.1 the $QpHpW$ value depends on the TPCH and TPCC which are the metrics of OLAP and OLTP respectively. $tpmC$ being *transactions per minute* is a metric of TPCC and $QphH$ being *Queries per hour* is the metric of TPCH.

This Unified HTAP metric is integrated into ottertune to be one of the target objectives. As this metric has two independent elements, optimization of the target objective can be done either by scalarization or by optimising both the independent elements individually.

### 3.4 Take-away:

This thesis focuses on improving the existing research in automated configuration tuning. To do the same, this work uses an existing solution [VAPGZ17] and has built the changes on it. [VAPGZ17] is an open source configuration tuning algorithm with highly stable and configurable software architecture. [VAPGZ17] being well documented
solution with git support, this system was chosen to further expand and build during this thesis.

In order to carry out the experiments and make the specified contributions in this thesis, five major changes were done to [VAPGZ17]. The major changes constitutes: addition of databases and versions, expansion of support for hybrid workloads, addition of scalarization techniques, integration of LHS sampling, and addition of multi-objective tuning agents. These changes are depicted in Figure 3.2.
4. Implementation and Experimental Setup

This chapter provides the details about the methods and models implemented. Starting from the databases and workloads, every change made is reasoned. This reasoning is explained in this chapter. With the information regarding the hardware and software characteristics, the reader is enabled to understand the system setup of the thesis through this chapter.

The structure of the chapter is as follows:

- **Databases and Workloads:** Section 4.1 describes the properties of databases used in the thesis. It also provides an overview of the standardised techniques for measuring the database performances.

- **Model Characteristics:** Section 4.2 provides the details and the properties of the models used in the thesis. It also helps the reader understand the reasoning of the choice of models and techniques.

- **Implementation Details:** Section 4.3 informs the user about the software and hardware characteristics of the thesis system architecture.

4.1 Databases and Workloads

This section brings light on the database management systems and the workloads that are added or extended. These changes are made
in order to support the study of this thesis. Among the databases, PostgreSQL is extended to support higher versions. This is explained in detail in Section 4.1.1. SingleStore (memSQL) is added to ottertune [VAPGZ17] so as to compare the results with the distributed database management system. This is detailed in Section 4.1.2. In order to support the booming hybrid workloads, HTAP is added to ottertune which is elaborated in the Section 4.1.3.

4.1.1 PostgreSQL

PostgreSQL is an open source database management system which supports object-relational databases. PostgreSQL being a row-based relational DBMS, theoretically, it is more suited for OLTP workloads. This DBMS ensures that the databases and transactions are ACID compliant.

Figure 4.1: postgreSQL Architecture Overview [Opr19]

Figure 4.1 provides an overview of the architecture of postgresQL. As can be seen in the architecture, PostgreSQL uses the disk memory as the storage manager.

PostgreSQL is chosen in this thesis as a DBMS to tune using ottertune for the following reasons:
4.1. Databases and Workloads

- Open source.
- Disk based memory architecture.
- Traditional relational database management system.
- Maintains ACID properties.
- Native to ottertune system.
- One of the stable and widely used DBMS.
- Conforms to SQL
- Supports multiple programming languages.

As known and repeated several times now, every database has different set of configuration knobs made available. As PostgreSQL is a traditional and stable DBMS, it provides wide range of knobs to be tunable by the users. Out of the available knobs, few were chosen to experiment in the thesis. The chosen knobs are listed in Table A.1.

The reason to reduce the number of knobs to tune is to reduce the search space of the ottertune. Even though ottertune and the supported agents are highly capable of tuning all the knobs, due to the time limit, the search space had to be reduced. The selected knobs are perceived to be the most impactful and easily modifiable. This is the sole intent of choosing this particular set of knobs to be tuned by ottertune during this thesis. More details on number of knobs for experiments is available in Chapter 5.

4.1.2 SingleStore (memSQL)

memSQL is a distributed relational database management system. But, unlike PostgreSQL, SingleStore (memSQL) supports both OLAP and OLTP workloads. It also comes with the support for in-memory architecture. Figure 4.2 shows the varied applications and workings of SingleStore (memSQL). SingleStore (memSQL) has some properties in common with PostgreSQL. These properties help us compare the two DBMS based on the properties on which they differ.

The reason to choose SingleStore (memSQL) for the experiments include both similarities and dissimilarities among PostgreSQL and SingleStore (memSQL). Similar to postgresQL, SingleStore (memSQL)
conforms to SQL. SingleStore (memSQL) transactions are ACID complaint and also supports multiple programming languages. The dissimilarities which among the two databases which help guage the diversity and robustness of ottertune are:

- Distributed system architecture.
- In-memory DBMS
- supports both OLAP and OLTP.
- proved to work with both TPC-C and TPC-H benchmarks [mem]
- non-native to ottertune.

Apart from the differences in terms of architecture of the DBMSes, some of the differences were noted while adding the SingleStore (memSQL) support to ottertune. Though these differences might not directly affect the performance of the DBMS, the execution and tuning steps were affected. These differences range from foreign keys, restart mechanisms and value range for the configuration knobs tuned. SingleStore (memSQL) does not support foreign keys unlike PostgreSQL.
4.2. Model Characteristics

To enforce any of the configuration changes on SingleStore (memSQL), all the distributed portion of memSingleStore (memSQL) SQL is restarted. It is also important to notice that any change to nearly all of the configuration knobs require a restart.

As mentioned in the previous section, for SingleStore (memSQL) as well, not all the knobs available for tuning are configured to be tuned by ottertune. The reason persists to be the same as to reduce the search space of the optimal configuration. The knobs chosen to tune are listed in Table A.2. More details on how these knobs contribute to the experiments is available in Chapter 5.

4.1.3 HTAP

Hybrid Transactional/Analytical Processing is a new type of workload which saw its advent with emergence of new business requirements. With more and more business systems requiring both analytical and transactional processing supported in the same database architecture, HTAP workloads and database architectures came into existence. HTAP workloads can have both analytical and transactional queries. As shown in Figure 4.3, HTAP properties are a combination of OLAP and OLTP properties.

Having said that, in order to measure the HTAP workload performance, Unified HTAP metric, proposed by [Paw19] is used. This metric is defined in detail in the section Section 3.3. In order to standardise the evaluation of HTAP workloads and database architectures, HTAP benchmarks are used. The benchmark used to evaluate database performance for HTAP workloads is proposed by [CPV+17]. Table 4.1 provides the list and description of the configurations for the benchmark.

This work is built upon the architecture of [VAPGZ17]. The architecture of ottertune primarily focused on tuning the database settings for OLAP and OLTP workloads. The support to tune database settings for HTAP workloads using the Unified HTAP metric is added to ottertune as part of this thesis.

4.2 Model Characteristics

This section draws light on the characteristics and properties of the models newly added to Otto-Tuner. Section 4.2.1, Section 4.2.2 and
4. Implementation and Experimental Setup

Figure 4.3: HTAP description [Har16]

<table>
<thead>
<tr>
<th>Workload Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target TPS</td>
<td>Indicates the desired value of transactions per second from the database. The workload bench ensures that this value is maintained by the OLTP transactions throughout the run of the workload. [CPV+17]</td>
</tr>
<tr>
<td>Warehouses</td>
<td>Specifies the overall size of the database.</td>
</tr>
<tr>
<td>OLTP Workers</td>
<td>The number of OLTP worker instances created by the workload in order to achieve the configured target tps value</td>
</tr>
<tr>
<td>OLAP Workers</td>
<td>Specifies the number of OLAP workers that will be increased in every new evaluation cycle.</td>
</tr>
<tr>
<td>Time</td>
<td>The duration for which the workload test will be run on the database.</td>
</tr>
<tr>
<td>Weights</td>
<td>Indicates the distribution of transaction types. Here, the distribution consists for both TPC-C and TPC-H transactions.</td>
</tr>
</tbody>
</table>

Table 4.1: HTAP workload benchmark parameters [CPV+17]

Section 4.2.3 provides the details of the models added to Otto-Tuner for sampling techniques, scalarization and multi-objective optimization.
4.2. Model Characteristics

4.2.1 Sampling techniques

Sampling techniques determine the criteria to select the point or a value from a given distribution. Understanding the sampling technique definition brings us to ask, where and why this is used in ottertune. Sampling techniques come into the picture when the DBA uses Neural networks or Gaussian Process Bandits as tuning agents for the session. As explained in Section 4.2.1, Once the agents are tuned, to select the next configuration based on the tuned model, sampling techniques are used. The flow chart in Figure 3.5 explaining the algorithm and the information flow for agents specifies in which exact step sampling techniques are used.

The sampling techniques used in this thesis are:

- **Random**: Random sampling technique works exactly as the name suggests. This technique selects the sample from a given distribution or the search space randomly. This is the most commonly used technique. The wide spread use of the techniques attributes to its ease and speed. Though the technique is widespread, it fails to provide consistency with the sample and search space coverage. This might lead to missing out on areas where there might be possible global minima or maxima. Random Sampling technique is the technique which is used by default in Otto-Tuner.

- **Latin Hyper Cube [Ima99]**: Latin Hyper cube sampling technique is a Stratified sampling method. This method focuses on covering all the parts of the search space while sampling. In other words, this method does not randomly cover the search space. The search space is distributed into equally probable sections. To form these sections, each of the dimension of the search space is divided into equal parts. From each of this section, a point or an item is chosen to form a sample. This is one of the well established techniques which is being used since years. Over the years, there have been many variants of this technique. One of the variants that is used this is work is Latin Hypercube Sampling with Multi-Dimensional Uniformity (LHSMDU) [Moz20]. LHSMDU helps provide more uniformity and supports high dimensional data.
4. Implementation and Experimental Setup

4.2.2 Scalarization methods

Scalarization is a technique that is used to convert problems with more than one target objectives to have a single target objective. This technique is mainly used when a single policy optimization has to be applied to multi-objective problem. Most of the solutions that are implemented for optimization focus on single target objective. This is the case even though most of the real world problems are not focused on only one target objective. Which in turn enforces the need to convert the multiple objective problems to be modified to fit into the mold of single policy optimization. This section provides an insight of the sampling techniques used to scalarize multiple objectives.

As part of this work, we have introduced HTAP workload to be tuned by ottertune system. In order to provide apt performance evaluation of HTAP workloads, Unified HTAP metric is inculcated. Unified HTAP metric is dependent on and directly proportional to two different factors. These two factors reflect the perform of OLAP and OLTP transactions within HTAP workload. This creates the need to optimise two different objectives at the same time. In other words, it is necessary to optimise both the factors of unified metric in order to optimise HTAP performance.

The tuning agents native to ottertune are single policy optimization techniques. In order to convert these techniques to support unified HTAP metric, following scalarization methods are used.

- **Chebyshev**: This method of scalarization reflects the value of each of the objective. Chebyshev method provides importance to the objectives proportional to its corresponding weights. To obtain the chebyshev scalarised value from all of the target objective values, one has to consider the maximum value among all of the weighted objectives. The method chebyshev Equation 4.1 implemented is obtained from [Bra18]. In Equation 4.1, the term $Obj_i$ refers to the $i^{th}$ target objective value. The term $Base_i$ refers to the default or initial value of $i^{th}$ target objective value.

$$Chebyshev \ = \ \max_{i \in [m]} (Obj_i - Base_i) \quad (4.1)$$
- **Weighted Rank-Sum**: Rank-Sum method uses the preferences of objectives to decide the ranks of the target objectives. Using these ranks, the weights for the objectives are calculated. The formula to calculate the weights from ranks is specified in Equation 4.2. This formula to calculate the weights for the objectives is obtained from [Gun18]. Rank-Sum weighting method is used to calculate the ranks of objectives in this thesis. Once the weights are calculated, the final scalarized Equation 4.3 value is calculated as the sum of product of weights and objective values. In Equation 4.3, $\text{Obj}_i$ refers to the $i^{th}$ target objective value.

\[
W_i = \frac{2(n + 1 - i)}{n(n + 1)} \tag{4.2}
\]

\[
\text{Rank - Sum} = \Sigma(W_i \times \text{Obj}_i) \tag{4.3}
\]

### 4.2.3 MO-MPO

Having introduced Unified HTAP metric to evaluate HTAP performance within ottertune. Unified HTAP metric is a multi-objective metric used for HTAP. The previous section (Section 4.2.2) has explained one of the ways in which multi-objective is handled in ottertune. This section elaborates how the other technique of multi-policy optimization is introduced in ottertune.

Multi-Objective Maximum a Posteriori Policy Optimization (MO-MPO) [AHH+20] is an algorithm combines re-enforcement learning and multi-objective optimization. In the scalarization techniques explained in Section 4.2.2, the two objectives are combined into a single value. This value is considered as a reward for the action taken by the policy. Hence, the two objectives contributing to the optimization has an impact in the reward space. In multi-objective optimization, the two objectives are not combined to form a single reward value. Rather, these objectives are considered separately in the calculation of policy distribution. And thus, in multi-objective optimization (MO-MPO) the multiple objectives are having an impact in the distribution space. In other words, Multi-objective optimization as opposed to single policy optimization does not convert the values of each of the objectives
into a single value reward. Each of the objective is considered separately for estimating the policy distribution.

MO-MPO is designed and mostly applied to Multi-Objective Markov Decision Processes (MO-MDP). The properties of MO-MDP such as states, actions, reward functions and transition probabilities can be easily mapped to the automated configuration tuning problem at hand. Here, the states and actions refer to the configuration settings provided by the algorithm, similar to choosing an action a new configuration is chosen in MO-MDP. Reward functions are the metric values of tpmC and QpqH. Transition probabilities indicates the probability that a configuration is chosen for the DBMS on the given workload. It in turn specifies the goodness of the configuration. These values are updated as and when the algorithm is tuned. This suggests that the probability of choosing same configuration settings (state or action) is reduced or increased with the metric (reward) of the configuration (action taken).

Apart from the similarities of the algorithm to the problem at hand, other reasons also contributed to the selection of this algorithm as a tuning agent. The option to specify the priorities of the objectives for the algorithm is one. It is not necessary for the algorithm to consider the objectives at hand with equal priorities. Also, the algorithm being reinforcement learning, can be fairly compared with the ottertune native algorithm Deep Deterministic Policy Gradient (DDPG) which uses single policy optimization.

Figure 4.4: Multi-Objective optimization by MO-MPO [AHH+20]
Figure 4.4 shows the high-level working of the algorithm [AHH+20]. In the figure, $S_0$ is the initial state. $S_U$ is the optimal state (Pareto front) which optimises both the objects with values $[3,3]$. This algorithm [AHH+20] tries to find the optimal solution, satisfying all the objectives, by finding the Pareto front based on the priorities of the objectives. In order to find the Pareto front, as MPO algorithm [AST+18] (explained in Section 2.3.3) is used, co-ordinate descent method is employed. KL-Divergence is measured to ensure that the estimated distribution is as close as possible to the actual distribution of configurations.

MO-MPO [AHH+20] is a policy iteration algorithm. Policy iteration means the policy of the model is updated iteratively using two steps: *policy evaluation* and *policy improvement*. *Policy evaluation* steps derives the Q-functions based on the policy of the model. *Policy improvement* uses the updated Q-functions to update the policy of the model. This process is repeated iteratively making this a policy iteration algorithm. It is to be noted that both Q-functions and policy are dependant on each other and optimising the other iteratively. This Q-function is maintained one for each of the objective being tuned. This whole process is explained in the algorithm Figure 4.5.

Here, we map the working of the algorithm to Otto-Tuner. For the first iteration, using the initial few observations (values of configurations and metrics) from the iterations of DBMS tuning session, initial policy is generated. This policy represents the state conditional distribution over action space of the configuration parameters (distribution of configurations and metrics). For the next iteration, this initial policy will be considered as an old policy. The old policy will be used to update the Q-functions (goodness of configuration) for every objective being optimised. From the updated Q-functions, old policy is updated to reflect the changes in Q-functions. Using the updated policy, which represents the state action distribution (distribution of configurations and metrics), the next action (configuration) is recommended by the algorithm.

4.3 Implementation details

This section details the characteristics of the device on which the experiments were conducted.
Algorithm 1 MO-MPO: One policy improvement step

1: given batch size ($L$), number of actions to sample ($M$), ($N$) Q-functions $\{Q_{k}^{\pi,old}(s, a)\}_{k=1}^{N}$, preferences $\{\epsilon_{k}\}_{k=1}^{N}$, previous policy $\pi_{old}$, previous temperatures $\{\eta_{k}\}_{k=1}^{N}$, replay buffer $\mathcal{D}$, first-order gradient-based optimizer $\mathcal{O}$

2: initialize $\pi_{\theta}$ from the parameters of $\pi_{old}$

3: repeat

4:   

5:     // Collect dataset $\{s^{i}, a^{i,j}, Q_{k}^{ij}\}_{i,j,k}^{L,M,N}$, where

6:     // $M$ actions $a^{i,j} \sim \pi_{old}(a|s^{i})$ and $Q_{k}^{ij} = Q_{k}^{\pi,old}(s^{i}, a^{i,j})$

7:     

8:     // Compute action distribution for each objective

9:     for $k = 1, \ldots, N$ do

10:        $\delta_{\eta_{k}} \leftarrow \nabla_{\eta_{k}} \eta_{k} \epsilon_{k} + \eta_{k} \sum_{i}^{L} \frac{1}{L} \log \left( \sum_{j}^{M} \frac{1}{M} \exp \left( \frac{Q_{k}^{ij}}{\eta_{k}} \right) \right)$

11:        Update $\eta_{k}$ based on $\delta_{\eta_{k}}$, using optimizer $\mathcal{O}$

12:        $q_{k}^{ij} \propto \exp \left( \frac{Q_{k}^{ij}}{\eta_{k}} \right)$

13:     end for

14:     

15:     // Update parametric policy

16:     $\delta_{\pi} \leftarrow -\nabla_{\theta} \sum_{i}^{L} \sum_{j}^{M} \sum_{k}^{N} q_{k}^{ij} \log \pi_{\theta}(a^{i,j}|s^{i})$ (subject to additional KL regularization, see Sec. 4.2.2)

17:     Update $\pi_{\theta}$ based on $\delta_{\pi}$, using optimizer $\mathcal{O}$

18: 

19: until fixed number of steps

20: return $\pi_{old} = \pi_{\theta}$

Figure 4.5: MO-MPO policy iteration algorithm. [AHH+20]
The experiments of this work are conducted on a single system to pro-
vide uniformity and to avoid the influence of the unknown variables 
on the results. The software and hardware characteristics of the sys-
tem is explained in the subsections: Section 4.3.1 and Section 4.3.2 
respectively.

4.3.1 Software

This section highlights the software properties of the system on which 
the experiments of this thesis were conducted. Further, this section 
also draws attention to the softwares and its versions installed in order 
to aid the working of OtterTune [VAPGZ17] software.

The major softwares used to conduct the experiments are as follows:

- **Operating System:** The operating system used is Linux 20.04. 
  This operating system is developer friendly. This also aids in easy 
  installation of other softwares.

- **Postgres DBMS:** One of the database management systems 
  being used is Postgres. Postgres DBMS is a native DBMS for ot-
terTune [VAPGZ17]. Postgres is a relational DBMS. The version 
  used for the experiments is 10.14.

- **SingleStore (memSQL) DBMS:** Another DBMS being used 
  in the experiments is SingleStore (memSQL) DBMS. SingleStore 
is inherently a distributed database management system. This 
DBMS supports both transactional and analytical workloads. The 
version of the DBMS used is 14.14.

- **Python:** Python is a scripting language in which the ottertune 
  [VAPGZ17] system is designed. The python version used in the 
  ottertune design is 3.6. For the study of this thesis, some changes 
to the ottertune [VAPGZ17] were made. These changes focused 
on accommodating more recent versions of ACME softwares. In 
order to aid that, version 3.8 of python is also installed. Currently, 
Otto-Tuner uses two versions of python - 3.6 and 3.8.

- **tensorFlow:** Tensorflow is a library which enables easy integra-
tion of AI and machine learning models. As Otto-Tuner focuses 
on AI in dbms, tensorflow plays an important role in the working 
of the system. The tensorflow version being used is 1.12.2.
• **deepmind/acme**: This is a library which focuses on implementation of latest reinforcement learning algorithms. As this work integrates a novel reinforcement learning algorithm into Otto-Tuner, acme library is being used. The version of the library being used is 0.2.0

• **oltpBench**: OLTP-Bench is a workload benchmarking tool. This provides a standard way of representing a workload and testing the performance of a DBMS for the specified workload. OLTP-Bench supports the benchmarking of both transactional and analytical workloads using TPC-C and TPC-H respectively. The version of this benchmarking tool being used is 1.0.

• **HTAPBench**: HTAPBench, similar to OLTP-Bench is a benchmarking tool. Unlike OLTP-Bench, HTAPBench helps standardise the testing of hybrid workloads. The version of this benchmarking tool being used is 1.0.

• **Java**: Java is one of the most important software that is supporting the working of OLTP-Bench, HTAPBench, and otter-tune. This helps interact with the DBMS client drivers and in turn connect to the DBMSes used in this thesis. The version of Java being used is 11.0.9.1.

• **Apache Maven**: Apache maven is the core software for the working on HTAPBench. HTAPBench uses this software to interact with the databases to execute the workloads and collect metrics. The version of maven being used is 3.6.3.

• **python django**: The otter-tune architecture is mainly built using python. Within python, django is used to fetch, update, insert and delete the records from MySQL database system. As otter-tune extensively uses these operations in every iteration of tuning process, django forms an irreplaceable library. The version of this library being used is 1.11.27.

4.3.2 Hardware

This work focuses on finding the optimal configuration which improves the database performance in an automated fashion. One major factor
which influences the database performance is the hardware characteristics on which the database system is being run. The same configuration on the same database against the same workload might not yield the same performance from the database. Hence, the characteristics of the machine on which the DBMS is run is of equal importance. This section focuses on mentioning the hardware characteristics of the machine on which all the experiments are run.

The characteristics of the hardware are specified in the Table 4.2.

<table>
<thead>
<tr>
<th>Component</th>
<th>Characteristics</th>
</tr>
</thead>
<tbody>
<tr>
<td>System vendor</td>
<td>Dell Inc.</td>
</tr>
<tr>
<td>CPU</td>
<td>Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz</td>
</tr>
<tr>
<td>CPU core and speed</td>
<td>The system supports 8 core CPUs with the minimum speed of 400.0000 MHz to maximum speed of 3400.0000 MHz. The number of kernel threads per core is 2.</td>
</tr>
<tr>
<td>RAM</td>
<td>8 Giga Bytes</td>
</tr>
</tbody>
</table>

Table 4.2: Hardware component characteristics of the experiment bed.

4.4 Outline:

This thesis has implemented the experiments on two different databases: Postgres and SingleStore (memSQL). Postgres is traditional a relational database whereas SingleStore (memSQL) is a distributed database. The reason to choose these two databases is to monitor the tuning difficulties on different database types.

This work analyses and compares the role and performance of multiple models of AI in configuration tuning. Few such models rely on sampling techniques to find a new configuration from the tuned models. Other than random sampling which was used native to [VAPGZ17], this work has integrate Latin hypercube sampling technique. This technique in theory provides higher and reliable coverage of the search space compared to random sampling.

Apart from that, the focus of this thesis has remained to expand the configuration tuning to support hybrid workloads and multi-objective optimizations. Hybrid workloads [Har16] are a combination of transactional and analytical queries on the database. The performance test
of the database for the hybrid workloads are done using HTAPBench [CPV+17].

In order to integrate hybrid workload tuning, [Paw19] metric needs to be supported by the tuning system. This is a multi-objective metric. This work uses ChebyShev [Bra18], Rank-Sum [Gun18] and simple multiplication techniques to scalarize the multi-objective result into the tuning system. Other than that, to avoid the loss of information during scalarization, multi-objective tuning agent is implemented.

MO-MPO (Multi-Objective MPO) is the agent which support multi-objective tuning. This is a RL algorithm which takes the values of each objective as different reward functions. This agent not only supports two objectives, instead the number of objectives can also be increased to be more than two.

Further, experiments are conducted on a computer with Intel i5 process, 8 cores and 8GB RAM. The operating system is Linux 20.08. Other softwares and libraries required for the execution of experiments are: Postgres DBMS, SingleStore (memSQL) DBMS, Python, TensorFlow, deepmind/acme, OLTPBench, HTAPBench, Java, Apache Maven and Python Django.
5. Evaluation

This chapter provides the details about the research questions and the experiments conducted to answer them. Further this chapter emphasizes the results and whether or not the results align with the expected outcome of the experiments.

The structure of the chapter is as follows:

- **Evaluation Overview:** Section 5.1 provides the reader with an overview of the criteria to evaluate the research questions. This section further details onto the reason behind the order of the experiments and how it contributes to the overall evaluation of the results.

- **Research Question 1: Efficiency of Exploration** Section 5.2 explains the first research question this thesis is focused on. Along with the list of experiments conducted, the results are presented to answer the question at hand.

- **Research Question 2: Influence of Sampling** Section 5.3 provides a detailed explanation of the research question 2, its focus and the list of experiments conducted. This section further shows the results of the experiments and its alignment towards the hypothesis.

- **Research Question 3: Multi-Objective Scalarization** Section 5.4 explores the influence of scalarization techniques. The list of experiments conducted in order to do so, is presented in this
section. This section further helps us answer which scalarization technique best suited the experiment conducted.

- **Research Question 4: Scalarization vs Multi-Objective Tuning** Section 5.5 focuses on optimization of multi-objective optimization without using scalarization techniques. Additionally, on comparing the results of the experiments conducted using multi-objective tuning agents with that of scalarization techniques.

### 5.1 Evaluation Overview:

This section focuses on providing the context for evaluation to the reader. It highlights the characteristics of the experiments, focus, evaluation criteria and the reason behind the order of experiments conducted.

To provide a clear view of the reasoning and focus behind the experiments, this section is further divided into the below subsections:

- **Experiment characteristics and commonalities:** Each experiment conducted for this thesis is run on the same computer. The reason to choose a single computer is to avoid the variation in performance due to variation in system characteristics. This helps to ensure the comparison is done entirely on the single variable that differs between the experiments.

  Further, to ensure uniformity, each experiment is run for a fixed number (150) of iterations. Every experiment starts with the default configuration of the DBMS. For every experiment is initiated the default configuration. The data generated from one model is not used by another model to train. The training opportunity provided for every model that is compared in this thesis is the same and remains fixed.

- **Evaluation criteria:** Evaluation criteria define how the experiments conducted in this thesis are evaluated. Experiments on three different workloads within this work. Each of these workloads comes with different properties and hence requires different metrics and benchmarks to measure its performance. The workloads considered are tabulated in ?? .
5.1. Evaluation Overview:

<table>
<thead>
<tr>
<th>Workload</th>
<th>Benchmark</th>
<th>Metric</th>
<th>Metric value</th>
</tr>
</thead>
<tbody>
<tr>
<td>OLTP</td>
<td>TPC-C</td>
<td>throughput</td>
<td>higher the better</td>
</tr>
<tr>
<td>OLAP</td>
<td>TPC-H</td>
<td>latency</td>
<td>lesser the better</td>
</tr>
<tr>
<td>HTAP</td>
<td>HTAPBench</td>
<td>Unified HTAP metric</td>
<td>higher the better</td>
</tr>
</tbody>
</table>

Table 5.1: Workload characteristics for experiments

For each of the workload mentioned in the Table 5.1, its corresponding benchmark and metric is used in every experiment the workload is involved. Among the 150 iterations run, the configuration which produces the maximum / minimum metric value is chosen as the best configuration produced by Otto-Tuner for that experiment.

- **Order of experiments:** This work groups the research questions into two. **Group1:** Research questions 1 and 2 focuses on finding the influence of models, knobs and sampling techniques on the optimal performance of the DBMS. **Group2:** Research questions 3 and 4 focuses on hybrid workload tuning and multi-objective optimization.

The reason for grouping is that, each of the research question is dependant on the other within a group. The results of the research question 1 is carried onto the research question 2. The same applies to research questions 3 and 4 where results from 3 is carried onto the research question 4. This enables time efficient evaluation of experiments.

Research question 1 executes the experiments with varied number of knobs. The set of knobs which perform the best is carried onto the further experiments. This is to reduce the redundancy in the experiments. Apart from this, the analysis of the experiments in **Group2** is made by comparing the results in research question 4 with that of the best value obtained in research question 3.

Because of the inter-relation between the research questions and the limited duration of the thesis, the experiments of research question 1 and 3 is conducted prior to that of research questions 2 and 4.

- **Understanding of the Results:** The experiments on Otto-Tuner are run for fixed number of iterations. For each of these
iterations, the Otto-Tuner simulates the workload being tested using the corresponding benchmark configured. Once the simulation is done, the metric and the configuration data is collected. Based on the collected information the tuning agent recommends the next configuration which can bring improvement in the performance. The performance of each of this iteration is put in terms of a graph. In this graph, y-axis shows the metric being evaluated and x-axis shows the number of the iteration as date and time. The best configuration for that tuning session is considered as the one with the highest value on y-axis (target metric). At the end of each iteration, the graph is scanned through to find the best performing configuration found by Otto-Tuner in the first 150 iterations.

5.2 Efficiency of Exploration

This research question focuses on finding the best performing exploratory agent. In Parallel, it also answers whether the number of knobs being tuned has any significance on the result or the optimal solution.

The hypothesis of the first research question of this thesis focuses on two things. First is to compare the performances of the different exploratory agents. Second, aims at discovering the relationship between the number of knobs tuned and the performance of the tuning session.

*Considering that the tuning sessions are run for a fixed number of iterations under the same experimental conditions, In comparison to the Deep Neural Network and Regression algorithms, Reinforcement Learning algorithm performs better by finding the better performing configuration setting.*

*Given the same tuning algorithm, and the tuning session being run for a fixed number of iterations under the same experimental conditions, higher number of knobs tuned, resulted in better improvement in the performance.*

The Section 5.2.1 explains the setup and provides the list of experiments that were run to answer this research question. Further, Section 5.2.2 provides the baseline for the conducted experiments against
5.2. Efficiency of Exploration

which the performance of the models are compared. The Section 5.2.3 provides the analysis of the results with respect to the baseline and the factors that influence the outcome.

5.2.1 Experiments and Setup

The experiments to analyse the influence of knobs and find out the better performing tuning agent are conducted on a single device to rule out the variations caused by the hardware. The characteristics of the device is detailed in the Section 4.3.

The experiments are conducted on two different database management systems: Postgres and SingleStore (memSQL). For these experiments, a single workload (OLTP) is considered. To simulate and evaluate OLTP workload, OLTPBenchmark [CPV+17] is used. The characteristic parameters of the benchmark are set to values according to Table A.3 and Table A.4 for SingleStore and Postgres respectively. The metric used to represent the performance of the workload is the number of transactions per second.

The experiments are conducted with three different sizes of the set of knobs: 10, 25 and 40. The goals of the experiments also remains to find the best performing knob set and the influence of the size on the performance of tuning. The knobs selected to tune are listed in the Table A.1 and Table A.2 for Postgres and SingleStore (memSQL) DBMS respectively. The tables Table A.1 and Table A.2 list the 40 knobs. When running experiments of 10 and 25 knobs, a subset of these knobs are considered. These subsets are selected in such a way that the knobs from all the resource groups present in the 40 knobs are reflected.

The tuning agents considered for comparison are: Gaussian Process Regression (GPR) model, Deep Neural Networks(DNN) and Deep Deterministic Policy Gradient(DDPG). As the name suggests, the GPR model is a regression model which tries to map the distribution of the points to a Gaussian process. Similar to that, DNN tries to emulate the points of the results obtained through a multi-layered neural network. Contrary to the other two models, DDPG is a reinforcement learning algorithm that uses an actor critic model to evaluate and find the next best configuration in the tuning process.
The Table 5.2 lists the group of experiments to answer the research question 1. The total number of experiments are 18 which are formed by the combination of models and databases for different number of knobs tuned mentioned in Table 5.2.

<table>
<thead>
<tr>
<th>Database</th>
<th>Workload</th>
<th>Knobs tuned</th>
<th>Model</th>
<th>Iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Postgres, Single-Store</td>
<td>TPC-C</td>
<td>10</td>
<td>GPR, DNN, and DDPG</td>
<td>150</td>
</tr>
<tr>
<td>Postgres, Single-Store</td>
<td>TPC-C</td>
<td>25</td>
<td>GPR, DNN, and DDPG</td>
<td>150</td>
</tr>
<tr>
<td>Postgres, Single-Store</td>
<td>TPC-C</td>
<td>40</td>
<td>GPR, DNN, and DDPG</td>
<td>150</td>
</tr>
</tbody>
</table>

Table 5.2: Experiments for evaluation of tuning models.

5.2.2 Expected Results

For the set of experiments conducted to answer research question 1, Model DNN (Deep Neural Networks) with 25 knobs are taken as a baseline. The Figure 5.2 and Figure 5.1 shows the execution result of DNN with 25 knobs tuned for OLTP workload on SingleStore and Postgres respectively.

![Figure 5.1](image)

Figure 5.1: DNN model performance on tuning 25 Postgres knobs for OLTP workload

From the Figure 5.2 and Figure 5.1, the best performing configuration value obtained are: 141 txn/sec for Postgres and 4542 txn/sec for SingleStore DBMS. These values are obtained by reading the graph as mentioned in Section 5.1

The performance of other models and different tuning knobs sizes are measured against these values.
5.2. Efficiency of Exploration

Apart from the throughput values, it is worth noting that DNN has a balance between the exploration and exploitation of the search space. After some iterations when the algorithm has explored a region of search space, it moves on to different regions of search space. This is reflected by the sudden rise or dip in the value of throughput in the graphs. And, there is no convergence at the end of 150 iterations. In order to know if the model would converge, higher number of iterations would provide more information.

The value ranges of both Postgres and SingleStore DBMS differ by a significant number. The overall performance of Postgres is in the range of hundreds whereas that of SingleStore in the range of thousands. This indicates the difference in the properties of these different DBMSes considered. This constitutes the main reason for the comparison not being made across databases to evaluate the performance of models and influence of the number of knobs.

Having said that, while comparing the models and analysing the results for first part of the hypothesis, we expect the DDPG model to perform better than DNN and GPR models. During comparison between the performances for different number of knobs, the expectation is to see the increase in performance with increasing number of knobs. This expectation regarding the number of knobs also aligns with the findings from [LZLG19].

5.2.3 Analysis

As mentioned, the focus of the research question 1 is to find the best performing model and the influence of the number of tuning knobs on
the performance. The Figure 5.3 and Figure 5.4 shows the results for Postgres and SingleStore DBMS respectively.

In Figure 5.3 and Figure 5.4, the values displayed in the dark blue column is the value of the DNN model. The DNN column for the 25 knobs is the value of the baseline considered for this research question.

**Models Comparison:** Considering the average performance of models for all the knob values, both DDPG and GPR models perform better than the DNN model. It is seen that the DDPG model performs better for Postgres and GPR performs better for SingleStore DBMS. The Table 5.3 shows the average performance gain in percentages for each of the model grouped by database management systems.

<table>
<thead>
<tr>
<th>Database</th>
<th>Model</th>
<th>Average Throughput gain (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SingleStore</td>
<td>DNN</td>
<td>1.16</td>
</tr>
<tr>
<td>SingleStore</td>
<td>DDPG</td>
<td>1.42</td>
</tr>
<tr>
<td>SingleStore</td>
<td>GPR</td>
<td>2.55</td>
</tr>
<tr>
<td>Postgres</td>
<td>DNN</td>
<td>0.20</td>
</tr>
<tr>
<td>Postgres</td>
<td>DDPG</td>
<td>0.05</td>
</tr>
<tr>
<td>Postgres</td>
<td>GPR</td>
<td>0.05</td>
</tr>
</tbody>
</table>

Table 5.3: Average Throughput gain based on model and database

From Table 5.3 it can be seen that, with just 150 iterations, the performance improvement is seen in all the cases even though it is not very significant. For SingleStore (memSQL), the gain percentage is higher compared to that of Postgres. Further, for SingleStore, the difference between the performance of DDPG and DNN is not significantly different. And for Postgres, DDPG and GPR has provided same average performances.

**Knobs Comparison:** Figure 5.3 and Figure 5.4 show a line which with plotted with the performance gain against the number of knobs tuned. The trend indicates that the performance increases with the increase in the number of knobs tuned. This applies to both Postgres and SingleStore (memSQL) databases.

This finding from the experiments is in line with the expected results. Using the results of this experiment, it can be said that the hypothe-
sis stating that *performance increases with the increase in number of knobs tuned* holds true.

![Figure 5.3: Comparison of model performance on Postgres DBMS](image)

![Figure 5.4: Comparison of model performance on SingleStore (memSQL) DBMS](image)

### 5.3 Influence of Sampling

This section focuses on experiments and results of the research question 2. i.e. to find if there is any influence of sampling techniques on the performance of the tuning models.

For Otto-Tuner, we are using two tuning agents which uses sampling techniques. These tuning agents are GPR and DNN. By default, OtterTune [VAPGZ17] using random sampling. Further, Latin Hypercube Sampling (LHS) technique is also implemented in Otto-Tuner.
This section focuses on finding which of the two: Random and LHS perform better if there is an influence of the sampling technique.

*When comparing the tuning sessions of LHS and random sampling, LHS performs better given that both the experiments are conducted on the same database, running on the same device / hardware with fixed tuning iterations.*

Latin Hyper-cube Sampling [Ima99] technique forms a grid of the search space and samples from each of the grid. This provides a wider coverage of the search space. Whereas, random sampling, as the name suggests samples randomly. This in turn can give rise to the possibility of uncovered regions in the search space.

With this major difference among the two sampling techniques, the technique which covers wider range of the search space is expected to perform better.

More details on this experiment and results are provided in further sections. Section 5.3.1 provides the details of the setup used for experiments and also lists the experiments that help answer the hypothesis of research question 2.

### 5.3.1 Experiments and Setup:

The experiments in this section focus on finding the influence of sampling techniques on the performance of tuning agent. To make sure there are less unknown and variables, all the experiments in this section are conducted on the same device with properties as mentioned in Section 4.3.

The experiments in this section are conducted on the databases: Postgres and SingleStore. Each of the experiment is run for fixed number of iterations of 150. For every experiments, 40 knobs are tuned. These knobs are listed in the Table A.1 and Table A.2. The reason for tuning 40 knobs are the results of the experiments for research question 1. The results in Section 5.2.3 show that with the increase in the number of knobs tuned, the performance improvement increases. Hence, 40 knobs are chosen for tuning.

The workloads used for the experiments in this section are: OLTP (Online Transaction Processing) and OLAP (Online Analytical Processing). OLTP uses the TPC-C for workload simulation and eval-
5.3. Influence of Sampling

The benchmark parameters are configured according to Table A.3 and Table A.4 for SingleStore and Postgres respectively. The metric used for OLTP performance evaluation is throughput which represents the transactions/second. This metric is tuned to find the maxima in the search space.

On the other hand, OLAP workload is simulated by TPC-H. TPC-H benchmark characteristics are set to values mentioned in Table A.5 and Table A.6 for SingleStore and Postgres DBMS respectively. This workload uses the metric latency which represents the time taken to execute a single query. This metric is tuned to find the minima in the search space. In other words, the aim of the tuner while tuning OLAP workload remains to find a configuration that takes minimum time to execute a query.

<table>
<thead>
<tr>
<th>Database</th>
<th>Workload</th>
<th>Model</th>
<th>Sampling Technique</th>
<th>Iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Postgres, Single-Store</td>
<td>TPC-C, TPC-H</td>
<td>GPR, DNN</td>
<td>LHS</td>
<td>150</td>
</tr>
<tr>
<td>Postgres, Single-Store</td>
<td>TPC-H</td>
<td>GPR, DNN</td>
<td>random</td>
<td>150</td>
</tr>
</tbody>
</table>

Table 5.4: Experiments to compare sampling techniques

The experiments conducted for this research question is listed in the Table 5.4. As the table suggests, two workloads and two sampling techniques are used. With the combination of workloads, databases, models and sampling techniques, total of 12 experiments are executed. The results of TPC-C workloads on LHS sampling for models GPR and DNN are reused from the experiments for research question 1. This does not bring any unknown variability as the experimental setup of the two remain the same.

5.3.2 Expected Results:

This research question compares the two sampling techniques. As random sampling technique is used by default, it is safe to say that the performance of LHS sampling technique is compared with that of random sampling technique. This makes the random sampling technique the baseline for this research question.

Section 5.2.3 shows that the GPR model performs better than DNN model. Due to the new findings, GPR model with random sampling
is updated to be the baseline for the experiments conducted in this section.

![Figure 5.5: GPR execution with random sampling on OLAP workload and Postgres DBMS](image)

![Figure 5.6: GPR execution with random sampling on OLAP workload and Single-Store DBMS](image)

According to the hypothesis, the experiments of this section should show a better performance of tuning with LHS sampling. When compared against the baseline for this section, the GPR model with LHS sampling is expected to show higher improvement than the tuning with random sampling. Similar comparison will be made for DNN model against LHS and random sampling. This comparison is also extended for two different workloads and databases.
5.3.3 Analysis:

The results of the experiments listed in Table 5.4 is shown in Figure 5.7 and Figure 5.8 for Postgres and SingleStore DBMS respectively.

Figure 5.7 and Figure 5.8 show the comparison between random and Latin hyper-cube sampling (LHS) techniques. These techniques are compared on two databases (Postgres and SingleStore), two workloads (OLAP and OLTP) using two tuning agents (DNN and GPR). In the figures, the x axis shows the agents used to tune the workload. The y-axes for the OLAP workloads indicate the improvement (decrease) in latency and for OLTP workload y-axes indicates the improvement (increase) in throughput.

\[
PI = \frac{(P_{\text{default}} - P_{\text{best}})}{100}
\]  

Equation 5.1

The y-axes which indicates the improvement of throughput and latency is calculated using the formula Equation 5.1. Where, PI indicates Performance Improvement, \(P_{\text{default}}\) refers to the throughput or latency of the default configuration for the DBMS and \(P_{\text{best}}\) indicates the values of throughput or latency for the best configuration found by the tuning agent for the respective DBMS.
Figure 5.8: Comparison of sampling techniques on SingleStore (memSQL) DBMS

From the results in Figure 5.7 and Figure 5.8, it can be seen that there is no influence of sampling technique for tuning OLAP workloads on SingleStore DBMS. However, in all other cases of tuning OLAP and OLTP on Postgres and OLTP on SingleStore DBMS, LHS technique has significantly made a difference.

The baseline for comparison in this section is GPR technique with random sampling. This baseline performance is exceeded by that of GPR Latin Hyper-cube Sampling technique. Hence, the results seen in Figure 5.7 and Figure 5.8 validates the hypothesis that the use of LHS provides better improvement in the tuning performance of agents compared to random sampling.

5.4 Multi-objective Scalarization

Another facet of this thesis is to explore the integration of hybrid workload in automated configuration tuning. In order to successfully integrate, multi-objective optimization is required. As this is not readily available in the existing tuning solutions, scalarization techniques are used.

This research question focuses on evaluating the scalarization techniques. In this thesis three different methods of scalarization: rank-sum [Gun18], Chebyshev [Bra18] and simple multiplication are used.
This section tries to answer which among these scalarization techniques would help capture the complete behaviour of the hybrid workload. In turn which technique would provide us a configuration which performs better. This is measured with the help of performance improvement compared to the default configuration setting of DBMS.

There exists no better performing scalarization technique that provides a higher improvement. All the scalarization techniques perform equally good in a set of experiments run on the same device and DBMS with fixed number of iterations to tune.

The hypothesis for this section suggests that there exists no better performing scalarization technique. It is possible that some of the techniques are better suited for some applications. But, it is unlikely that one technique is always better than the other.

This section attempts to prove or disprove the hypothesis stated for research question 3. Section 5.4.1 explains the setup and provides the list of experiments conducted to answer the research question 3. Further, Section 5.4.2 provides an overview of the expected behaviour of the experiments. Following which, Section 5.4 compares the actual results against the expected behaviour.

5.4.1 Experiments and Setup:

For the experiments, two databases: Postgres and SingleStore (memSQL) are used. The experiments in this section are conducted on the hybrid workload on both the database systems. For the hybrid workload, HTAPBench [CPV+17] is used to emulate the workload. The HTAPBench configuration is listed in Table A.7 and Table A.8 for SingleStore and Postgres respectively. As mentioned in [Paw19] Unified HTAP metric is used to evaluate the performance of the hybrid workload.

This metric comprises of two components.

- $tmpC$ (Transactions per minute): that evaluates the performance of the DBMS against transactional queries.
- $QpqH$ (Queries per hour): evaluates the performance of the DBMS against the analytical queries.
HTAPBench provides these two components as the metric to evaluate the DBMS performance. These two components are scalarized within Otto-Tuner to enable the tuning agents train and suggest the next configuration. This section evaluates if all the scalarization techniques perform the same.

<table>
<thead>
<tr>
<th>Database</th>
<th>Workload</th>
<th>Scalarization Technique</th>
<th>Model</th>
<th>Iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Postgres, Single-Store</td>
<td>HTAP</td>
<td>Rank-Sum</td>
<td>DDPG</td>
<td>150</td>
</tr>
<tr>
<td>Postgres, Single-Store</td>
<td>HTAP</td>
<td>ChebyShev</td>
<td>DDPG</td>
<td>150</td>
</tr>
<tr>
<td>Postgres, Single-Store</td>
<td>HTAP</td>
<td>Simple Multiplication</td>
<td>DDPG</td>
<td>150</td>
</tr>
</tbody>
</table>

Table 5.5: Experiments to compare scalarization techniques.

Table 5.5 provides the list of experiments conducted to support the analysis for research question 3. Each of the three scalarization techniques are tested against the two databases used in the experiments. This leads to total of 6 experiments being conducted.

5.4.2 Expected Results:

In this set of experiments, the only variable is the scalarization technique. The techniques used: Rank-sum [Gun18], chebyshev [Bra18] and simple multiplication has the same source for scalarization. In other words, the workload, workload benchmark, tuning agent, databases and iterations remain the same. When only the scalarization technique differs, theoretically, the results should not vary from one another significantly.

According to the hypothesis of the research question being answered in this section and the considered variables in the experiment, all the scalarization techniques provide equally optimal solution. Hence, the theoretical expectation from these experiments remain that there would be no single better scalarization technique.

Nevertheless to unbiasedly analyse the results of the experiments, this section specifies a baseline for the comparison. Simple multiplication scalarization technique multiplies the values of tpmC and QpqH and further divides the value with the number of OLAP workers configured
within the HTAPBench. This technique is straightforward and has minimal transformation of the two components that are being scalarized. The performances of the DDPG model on the HTAP workload using the scalarization methods Rank-sum [Gun18] and chebyshev [Bra18] is compared with that of simple multiplication scalarization.

Also, the DBMS SingleStore (memSQL) inherently supports the performance of the hybrid workload better. This section conducts the experiments on hybrid workloads, for which, the SingleStore database is considered as a baseline with the scalarization of simple multiplication. This baseline having the better support for hybrid workloads, is expected to show better performance improvement than for the experiments conducted on Postgres database.

5.4.3 Analysis:

The experiments in this section help us find out which scalarization provides better results or if all the techniques perform equally. The results of the experiments in this section is outlined in Figure 5.9.

![Comparison of scalarization techniques](image.png)

Figure 5.9: Comparison of scalarization techniques

Figure 5.9 shows the best performance value obtained on the six experiments conducted. The x-axis has the experiments grouped based on databases. y-axis indicates the best performance value obtained for the corresponding experiment.
Comparison of database performance on hybrid workloads: From Figure 5.9, it can be seen that the SingleStore (memSQL) database performs better with the techniques: chebyshev and rank-sum. On the other hand, for simple multiplication, Postgres performs better compared to SingleStore even though Postgres is not as highly compatible with hybrid workloads as the SingleStore.

Figure 5.10: Execution of HTAP workload on SingleStore DBMS using simple multiplication scalarization

Figure 5.11: Execution of HTAP workload on Postgres DBMS using simple multiplication scalarization

Comparison of scalarization techniques: Figure 5.9 shows that simple multiplication has provided better results for both the databases considered in the experiments. Even though the performance is not as expected for the database SingleStore, it exceeds the performances of other scalarization techniques for SingleStore. This finding is not in sync with the hypothesis of the research question 3. The results obtained indicate there exists an influence of the scalarization technique. It is unaware if this holds true for other workloads, but for
the hybrid workload, simple scalarization performs better compared to other scalarization techniques.

Figure 5.10 and Figure 5.11 show the exploratory path of the DDPG tuning agent using simple scalarization technique for SingleStore and Postgres respectively. From the figures, it can be seen that there has been higher amount exploration compared to exploitation of the search space. Both the algorithms do not show signs of convergence at the end of 150 iterations. According to the figures, the trend of performance is not nearby the best performance found for each of the experiments.

5.5 Scalarization vs Multi-Objective tuning

The integration of hybrid workload with the automated configuration tuning is relatively new. That being the reason, none of the existing automated configuration tuning solutions support hybrid workload to the best of the author’s knowledge. Hence, there exist least to no solutions that support multi-objective optimization in the configuration tuning.

The previous research question (Section 5.4) focuses on converting the result of the hybrid workload to be suited within the existing solutions. On the contrary, the current research question focuses on modifying or expanding the existing solutions to support the tuning agents which can optimise multiple objectives simultaneously. This is achieved by the use of multi-objective MPO [AHH+20] tuning agent in Otto-Tuner.

**In comparison with the scalarization techniques, multi-objective tuning agents provide better results, provided that the comparison is made on the same DBMS run on the same system characteristics for a fixed number of iterations.**

Now that there exist two ways to expand the support of automated configuration tuning to hybrid workloads, it is also necessary to know which method is better. The research question in this section tries to answer the same. The experiments conducted focus on comparing the performance of scalarization techniques with that of multi-objective optimization. The results of the experiments in this section helps answer if the hypothesis of this section specifies hold true or is disproved.
Section 5.5.1 describes the experimental setup used to conduct the experiments for this research question. This section also lists the experiments conducted to prove or disprove the hypothesis. Further Section 5.5.2 provides the context to analyse the results of the experiments. This section specifies the comparison baseline used to analyse the experimental results of this research question. Followed by Section 5.5.3 concludes the research question by analysing the results. The results of the experiments are compared against the baseline specified and justified in this section.

5.5.1 Experiments and Setup:

The experiments for research question 4 focuses on tuning database configuration to support hybrid workloads. Hence, the model used for tuning is Multi-Objective MPO [AHH+20]. This agent tunes two different databases: Postgres and SingleStore (memSQL). The experiments are run for fixed number of iterations of 150. Both the experiments are run on the same device. The device characteristics are specified in detail in Section 4.3.

Hybrid workload is simulated and evaluated using the HTAPBench [CPV+17]. HTAPBench parameter values used for this set of experiments is mentioned in Table A.7 and Table A.8 for SingleStore and Postgres DBMS respectively. Metric used to evaluate the performance of the DBMS against the hybrid workload is the Unified HTAP metric [Paw19]. As mentioned earlier, this metric consists of two components: tpmC and QpqH. The value of the DBMS performance (Unified HTAP metric) was sent to the tuning agent as a single value after scalarization in the previous experiments Section 5.4. It is worth noting that, in this section of experiments, the values for these components are sent to the agent to tune as two separate values. The tuning agent trains to optimise these values separately. Based on the tuned agent model, the next configuration is specified. This in turn helps in finding the optimal configuration setting for the DBMS to support hybrid workloads.

Table 5.6 lists the experiments conducted for research question 4. As the table enlists, a total of two experiments are conducted. One experiment for each database used in the experimental analysis. For each of the experiments, hybrid workload and Multi-Objective MPO (MO-MPO) [AHH+20] tuning agent is used.
5.5. Scalarization vs Multi-Objective tuning

5.5.2 Expected Results:

The experiments for research question 4 are conducted on multi-objective MPO [AHH+20] tuning agent. The motivation to use MO-MPO tuning agent is to support hybrid workload tuning. Scalarization techniques experimented on in the previous research question (Section 5.4) also focus on integrating hybrid workloads in tuning. But, it is not known as to which method performs better.

The comparison between the performance of MO-MPO and that of scalarization techniques helps draw insight to this research question. MO-MPO is a single technique and there exists multiple scalarization techniques on which experiments were conducted. For this reason, the scalarization technique that performs the best among the compared techniques will be used as a baseline for the comparison in this section. As the hypothesis in Section 5.4 suggests, there is a possibility that all the techniques perform equally good. In such a case, the scalarization technique which acted as a baseline for comparison: simple multiplication, will be used as a baseline in this section.

This in turn leads to the comparison between DDPG and MO-MPO technique. Both DDPG and MO-MPO techniques are reinforcement learning agents with different characteristics. These tuning agents are explained in detail in the Section 2.3.2 and Section 4.2.3 respectively. Even though MO-MPO supports multi-objective tuning, it can also be used for single objective tuning. Thus, this section provides a comparison between the baseline/best performing scalarization technique using DDPG against the MO-MPO tuning agent.

5.5.3 Analysis:

Total of two experiments were conducted in this section to compare the performance of individual optimization of multiple objectives against the performance of scalarized optimization of multiple objectives.

The results of the two experiments are shown in Figure 5.12. The bars marked with mustard colour indicates the value of scalarized tuning
session. The bars with the blue indicate the performance of MO-MPO. The results of MO-MPO is maintained as a two component value. To compare the value with the simple multiplication scalarized value, the performance value of MO-MPO is also scalarized using simple multiplication. It is to be noted that this conversion is only for comparison and not done during the tuning of the configurations.

Figure 5.12: Comparison of scalarization with Multi-Objective Optimization

**Scalarization vs Multi-Objective Optimization:** Figure 5.12 shows the results as well as the comparison between the databases and the multiple objective optimization methods. y-axis indicates the percentage improvement provided by each of the experiments with respect to the default configuration.

In contrast to the expected outcome, the performance improvement of the Postgres DBMS is higher compared to that of SingleStore. The SingleStore being inherently supportive of hybrid workloads, the default performance of the DBMS itself would be higher. This can be considered as a cause for the lower improvement in performance. In other words, Postgres being not designed to support hybrid workloads, there is a lot of room for improvement by tuning the configuration values.

It is seen that optimising multiple objectives individually has performed better compared to the scalarized tuning session. With mul-
tiple objective optimization (MOO), there is modification or trans-
formation no of the values seen from the database metrics. This in
turn suggests there is no information loss between the values and the
rewards specified for the tuning agents. With respect to the hypothe-
sis for this section, MOO has performed better than the scalarization
techniques. Thus satisfying the hypothesis.

Figure 5.13: Execution of Mo-MPO on SingleStore DBMS for HTAP workload

Figure 5.14: Execution of Mo-MPO on Postgres DBMS for HTAP workload

Figure 5.13 and Figure 5.14 shows the exploratory path of the MO-
MPO on hybrid workloads on SingleStore and Postgres respectively.
Figure 5.13 shows that the SingleStore DBMS performance has iterations with performance value 0. This anomaly reflects the behaviour of
Otto-Tuner rather than the performance of MO-MPO. This anomaly
is further explained in Section 6.1.

In terms of exploration of MO-MPO, better exploitation is seen com-
pared to that of DDPG in Section 5.4.3. The exploration is higher
compared to exploitation and the convergence is not seen similar to
5. Evaluation

DDPG. Also similar to DDPG, the convergence or the exploitation is least to none near the once found best performing configuration value.

5.6 Summary:

This chapter evaluates and analyses the performances of multiple AI models in the automated configuration tuning solution Otto-tuner. Having found that the performance of reinforcement algorithm and regression methods perform better, further comparison is made with the sampling techniques. As expected, among the sampling techniques random sampling which is highly inconsistent is outperformed by Latin Hyper-cube Sampling. With the use of results conducted for research question 3, simple multiplication is found to be the best performing scalarization technique. The results of this technique is compared with that of MO-MPO tuning agent. With this comparison it can be concluded that the MO-MPO performs better than any other scalarization.
6. Conclusions and Future Work

As with any solution or an idea in research, Otto-Tuner has its drawbacks. This chapter concludes the thesis by providing an evaluation of Otto-Tuner in Section 6.1 and the possible future works on Otto-Tuner in Section 6.2.

6.1 Otto-Tuner: Pros and Cons

Otto-Tune is an extension of OtterTune \cite{VAPGZ17}. The goal of Otto-Tuner remains the same of OtterTune. Otto-Tuner aims to provide automation for configuration tuning and alleviate the strenuous procedure of manual tuning.

Otto-Tuner has attempted to improve the automation of configuration tuning by supporting hybrid workloads and multi-objective optimization. In spite of the contributions of Otto-Tuner, it has to be evaluated so that it can be improved. This section details the good aspects and the bad of Otto-Tuner.

The good aspects of Otto-Tuner are as follows.

- **Flexibility:** Otto-Tuner is flexible in terms enabling addition of new databases, workloads and metrics. Otto-Tuner is highly configurable to enable support for new databases and versions. This makes the solution versatile and robust.

- **User Interface:** Otto-Tuner comes with the web interface which can used by DBA. For the DBA to use the web interface,
minimal knowledge of the tuning agents and the architecture of Otto-Tuner is required.

- **Choice of knobs:** Otto-Tuner allows the DBA to be able to decide which knobs can be tuned and which needs to be constant. Otto-Tuner comes with the preset knobs which are tunable by default for a selected DBMS. If there arises a case where the DBA finds some other knobs needs tuning or some don’t, this preset knobs can be modified to reflect the needs of the DBA.

The aspects of the Otto-Tuner that needs improvement are as follows.

- **Inconsistent Behaviour:** During the execution of Otto-Tuner, for each iteration of execution, workload benchmark tool simulates the workload to evaluate the DBMS performance. The metric obtained that represents the performance of the DBMS is not highly consistent. In other words, the throughput or the latency value obtained for a given configuration of DBMS can vary within a certain range. This makes the Otto-Tuner a bit less reliable.

  The source of this inconsistency can be the performance of the DBMS against the benchmark workload or the Otto-Tuner or both. To gain highly reliable performance from the DBMS, this problem needs to be solved. Every solution seen for the conducted experiments needs to be highly reproducible.

- **Iteration Gamble:** Otto-Tuner runs for iterations to train the model used for tuning and provide an optimal solution. This brings us to the question as to how many iterations of experiments should be run to get the most optimal solution of configurations? How many number of iterations help train the model without over-fitting and under-fitting the model?

  DBA needs to answer these question every time DBA chooses to tuning the configurations using Otto-Tuner. DBA might not be well versed to answer these questions which in turn leads to confusion and poorly executed experiments with bad results. To avoid having bad results due to wrong number of iterations, this problem needs to be solved.

- **Invalid Configuration:** The knobs that are being tuned by Otto-Tuner are inter-related. Otto-Tuner trains the models to
learn this inter-relations between knobs. In the mean while, when a value is set for these inter-related knobs that doesn’t bode well with the DBMS configuration, DBMS crashes. These configuration having the invalid values for the knobs causes the DBMS to crash and in turn halt the tuning process.

This is one the biggest cons of the Otto-Tuner. Receiving the feedback from the DBMS and flagging the configuration as invalid theoretically helps tune the model to provide a better configuration and hence avoid further invalid configuration values. A solution to this problem enables Otto-Tuner to work in a more robust way.

- **Needle in a hay sack:** Otto-Tuner runs the tuning session for a specified number of iterations. In this experiment, the fixed number of iterations are 150. Once these iterations are run, Otto-Tuner doesn’t suggest if it has found the best configuration or not. The DBA needs to go through the graph presented in the web interface to find the best configuration.

   Each configuration within the graph is represented as a point. When the mouse is hovered over the point, the metric value of the configuration represented by the point is displayed. When the multiple points have similar metric value, this point becomes clustered to view. Thus making the readability minimum. Figure 5.5 is an example of such a graph. In this picture, the iterations are 150. If the number of iterations are more, points become more clustered. Thus making the graph illegible and hence the found optimal configuration becomes undiscovered.

### 6.2 Future Works

Otto-Tuner focused on improvement of automated configuration tuning. In the process some of the aspects like hybrid workload and multi-objective optimization are achieved. Some of the aspects like time series based evaluations were not possible to be made in this thesis due to time constraints. The ideas that could not be explored in this thesis but can be improve the performance of Otto-Tuner are listed in this section as follows.
• **Temporal Aspects:** Otto-Tuner collects the metrics of the DBMS on a workload to evaluate the performance for a given configuration. This metric is a single number representing the all the variations in the DBMS throughout the duration of the workload execution on DBMS. This single value provides an average view of the DBMS behaviour for the workload over a period of the execution.

During a given execution period, the performance of the workload may vary for every unit time. This variation of the performance with respect to the time aspect is not captured by Otto-Tuner. To capture these variation in DBMS behaviour, time series data can be used. To create time series data, DBMS needs to be probed to collect the metrics at a certain interval. The metrics collected at an interval forms the time series representing the DBMS behaviour while the workload was executing on the DBMS.

This obtained time series can be used by Otto-Tuner to obtain better configuration settings for a database on a given workload.

• **More exploratory agents:** Today in Otto-Tuner, four kinds (Regression model, Deep Neural Networks, Single objective reinforcement learning and multi-objective reinforcement learning) of tuning agents are used. This group of tuning agents can be further extended. The extension of tuning agents can also be focused on providing support for time series based tuning of configurations.

• **Otto-Tuner+:** Section 6.1 lists the aspects of the Otto-Tuner that hinder the performance of Otto-Tuner and needs improvement. The solution for these problems on Otto-Tuner would provide a better solution that tunes configuration in an automated fashion. Otto-Tuner+ would overcome these problems and provide highly reliable solution.

### 6.3 Concluding Remarks

This thesis extends the automated configuration tuning solution OtterTune [VAPGZ17] to Otto-Tuner. Otto-Tuner expands the ottertune to support SingleStore (memSQL) DBMS, Hybrid workloads and metrics and multi-objective optimizations. As part of this thesis, analysis
of the tuning agents currently used in the configuration tuning solutions are made. From the comparison, it is evident that reinforcement learning algorithm provides a sustainable growth for the improvement in the tuning process. Among the sampling techniques, even though random sampling provides good solutions at times, it proves to be unreliable. To adapt to the real world needs, multi-objective optimization is necessary. This can be achieved with scalarization. On the other hand, optimising each of the objectives individually provides a better solution. The automated configuration tuning can be extended for improved performance using time series representation of the DBMS behaviour for a workload.
6. Conclusions and Future Work
A. Appendix
<table>
<thead>
<tr>
<th>Configuration Knobs tuned for Postgres</th>
</tr>
</thead>
<tbody>
<tr>
<td>autovacuum</td>
</tr>
<tr>
<td>archive_mode</td>
</tr>
<tr>
<td>effective_cache_size</td>
</tr>
<tr>
<td>maintenance_work_mem</td>
</tr>
<tr>
<td>max_wal_size</td>
</tr>
<tr>
<td>max_worker_processes</td>
</tr>
<tr>
<td>sharedBuffers</td>
</tr>
<tr>
<td>temp_buffers</td>
</tr>
<tr>
<td>wal_buffers</td>
</tr>
<tr>
<td>work_mem</td>
</tr>
<tr>
<td>bgwriter_delay</td>
</tr>
<tr>
<td>bgwriter_flush_after</td>
</tr>
<tr>
<td>bgwriter_lru_maxpags</td>
</tr>
<tr>
<td>bgwriter_lru_multiplier</td>
</tr>
<tr>
<td>dynamic_shared_memory_type</td>
</tr>
<tr>
<td>geqo</td>
</tr>
<tr>
<td>geqo_effort</td>
</tr>
<tr>
<td>lock_timeout</td>
</tr>
<tr>
<td>max_locks_per_transaction</td>
</tr>
<tr>
<td>max_parallel_workers_per_gather</td>
</tr>
<tr>
<td>max_pred_locks_per_transaction</td>
</tr>
<tr>
<td>max_prepared_transactions</td>
</tr>
<tr>
<td>wal_compression</td>
</tr>
<tr>
<td>wal_keep_segments</td>
</tr>
<tr>
<td>wal_writer_delay</td>
</tr>
<tr>
<td>vacuum_cost_limit</td>
</tr>
<tr>
<td>track_activity_query_size</td>
</tr>
<tr>
<td>replacement_sort_tuples</td>
</tr>
<tr>
<td>old_snapshot_threshold</td>
</tr>
<tr>
<td>tcp_keepalives_interval</td>
</tr>
<tr>
<td>tcp_keepalives_count</td>
</tr>
<tr>
<td>transaction_deferrable</td>
</tr>
<tr>
<td>track_commit_timestamp</td>
</tr>
<tr>
<td>synchronous_commit</td>
</tr>
<tr>
<td>synchronize_seqscans</td>
</tr>
<tr>
<td>track_io_timing</td>
</tr>
<tr>
<td>track_functions</td>
</tr>
<tr>
<td>track_counts</td>
</tr>
<tr>
<td>wal_sync_method</td>
</tr>
<tr>
<td>vacuum_cost_page_miss</td>
</tr>
</tbody>
</table>

Table A.1: Chosen knobs to tune for PostgreSQL
<table>
<thead>
<tr>
<th>Configuration Knobs tuned for SingleStore (memSQL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>load_data_read_size</td>
</tr>
<tr>
<td>read_advanced_counters</td>
</tr>
<tr>
<td>workload_management_max_queue_depth</td>
</tr>
<tr>
<td>workload_management_max_connections_per_leaf</td>
</tr>
<tr>
<td>sp_query_dynamic_param</td>
</tr>
<tr>
<td>workload_management</td>
</tr>
<tr>
<td>load_data_write_size</td>
</tr>
<tr>
<td>load_data_internal_compression</td>
</tr>
<tr>
<td>default_distributed_ddl_timeout</td>
</tr>
<tr>
<td>multi_insert_tuple_count</td>
</tr>
<tr>
<td>resource_governor_cpu_limit_mode</td>
</tr>
<tr>
<td>columnstore_flush_bytes</td>
</tr>
<tr>
<td>workload_management_enable_static_partitioning</td>
</tr>
<tr>
<td>workload_management_queue_timeout</td>
</tr>
<tr>
<td>lock_wait_timeout</td>
</tr>
<tr>
<td>highlight_fragment_size</td>
</tr>
<tr>
<td>workload_management_max_threads_per_leaf</td>
</tr>
<tr>
<td>pipelines_batches_metadata_to_keep</td>
</tr>
<tr>
<td>workload_management_memory_queue_threshold</td>
</tr>
<tr>
<td>workload_management_memory_queueuming</td>
</tr>
<tr>
<td>resource_usage_model</td>
</tr>
<tr>
<td>columnstore_segment_rows</td>
</tr>
<tr>
<td>default_autostats_enabled</td>
</tr>
<tr>
<td>data_conversion_compatibility_level</td>
</tr>
<tr>
<td>node_degree_of_parallelism</td>
</tr>
<tr>
<td>max_connections</td>
</tr>
<tr>
<td>max_connection_threads</td>
</tr>
<tr>
<td>query_parallelism</td>
</tr>
<tr>
<td>interpreter_mode</td>
</tr>
<tr>
<td>backup_max_threads</td>
</tr>
<tr>
<td>autocommit</td>
</tr>
<tr>
<td>auditlog_rotation_size</td>
</tr>
<tr>
<td>change_count_write_interval</td>
</tr>
<tr>
<td>enable_binary_protocol</td>
</tr>
<tr>
<td>enable_multipartition_queries</td>
</tr>
<tr>
<td>ignore_insert_into_computed_column</td>
</tr>
<tr>
<td>load_data_max_buffer_size</td>
</tr>
<tr>
<td>max_connect_errors</td>
</tr>
<tr>
<td>optimize_stmt_threshold</td>
</tr>
</tbody>
</table>

Table A.2: Chosen knobs to tune for memSQL
### TPC-C Workload Bench Parameter Values

<table>
<thead>
<tr>
<th>Workload Parameters</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scale Factor</td>
<td>2</td>
</tr>
<tr>
<td>Terminal</td>
<td>2</td>
</tr>
<tr>
<td>Time</td>
<td>10 seconds</td>
</tr>
<tr>
<td>Rate</td>
<td>10000 txns/sec</td>
</tr>
<tr>
<td>Weights</td>
<td>45,43,4,4,4</td>
</tr>
</tbody>
</table>

Table A.3: TPC-C workload bench parameter values for SingleStore (memSQL)

### TPC-C Workload Bench Parameter Values for Postgres

<table>
<thead>
<tr>
<th>Workload Parameters</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scale Factor</td>
<td>1</td>
</tr>
<tr>
<td>Terminal</td>
<td>1</td>
</tr>
<tr>
<td>Time</td>
<td>5 seconds</td>
</tr>
<tr>
<td>Serial</td>
<td>True</td>
</tr>
<tr>
<td>Rate</td>
<td>unlimited</td>
</tr>
<tr>
<td>Weights</td>
<td>0,1,0,0,0,0,1,0,0,0,0,0,0,0,1,1,0,0,0,0,1</td>
</tr>
</tbody>
</table>

Table A.4: TPC-C workload bench parameter values for Postgres

### TPC-H Workload Bench Parameter Values

<table>
<thead>
<tr>
<th>Workload Parameters</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scale Factor</td>
<td>1</td>
</tr>
<tr>
<td>Terminal</td>
<td>1</td>
</tr>
<tr>
<td>Time</td>
<td>5 seconds</td>
</tr>
<tr>
<td>Serial</td>
<td>True</td>
</tr>
<tr>
<td>Rate</td>
<td>unlimited</td>
</tr>
<tr>
<td>Weights</td>
<td>0,1,0,0,0,0,1,0,0,0,0,0,0,0,1,1,0,0,0,0,1</td>
</tr>
</tbody>
</table>

Table A.5: TPC-H workload bench parameter values for SingleStore (memSQL)

### TPC-H Workload Bench Parameter Values for Postgres

<table>
<thead>
<tr>
<th>Workload Parameters</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scale Factor</td>
<td>1</td>
</tr>
<tr>
<td>Terminal</td>
<td>1</td>
</tr>
<tr>
<td>Time</td>
<td>5 seconds</td>
</tr>
<tr>
<td>Serial</td>
<td>True</td>
</tr>
<tr>
<td>Rate</td>
<td>unlimited</td>
</tr>
<tr>
<td>Weights</td>
<td>0,1,0,0,0,0,1,0,0,0,0,0,0,0,1,1,0,0,0,0,1</td>
</tr>
</tbody>
</table>

Table A.6: TPC-H workload bench parameter values for Postgres
<table>
<thead>
<tr>
<th>Workload Parameters</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target TPS</td>
<td>5</td>
</tr>
<tr>
<td>Warehouses</td>
<td>35</td>
</tr>
<tr>
<td>OLTP Workers</td>
<td>5</td>
</tr>
<tr>
<td>OLAP Workers</td>
<td>5</td>
</tr>
<tr>
<td>Time</td>
<td>5 seconds</td>
</tr>
<tr>
<td>Weights</td>
<td>45,44,4,4,3,3,2,3,2,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5</td>
</tr>
</tbody>
</table>

Table A.7: HTAP workload bench parameter values for SingleStore (memSQL)

<table>
<thead>
<tr>
<th>Workload Parameters</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target TPS</td>
<td>5</td>
</tr>
<tr>
<td>Warehouses</td>
<td>35</td>
</tr>
<tr>
<td>OLTP Workers</td>
<td>5</td>
</tr>
<tr>
<td>OLAP Workers</td>
<td>5</td>
</tr>
<tr>
<td>Time</td>
<td>5 seconds</td>
</tr>
<tr>
<td>Weights</td>
<td>45,44,4,4,3,3,2,3,2,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5</td>
</tr>
</tbody>
</table>

Table A.8: HTAP workload bench parameter values for Postgres
Bibliography


IEEE Transactions on Knowledge and Data Engineering, PP:1–1, 05 2020. (cited on Page 10)
