UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

NICOLAU PEREIRA ALFF

Learned Indexes with Updates

Monografia apresentada como requisito parcial para a obtenção do grau de Bacharel em Ciência da Computação

Orientador: Prof[a]. Dr[a]. Renata Galante

Porto Alegre
2023

AGRADECIMENTOS

Inicialmente, gostaria de expressar meu imenso agradecimento à minha família: meus pais - Marco e Maira - minha irmã - Hannah - e minha avó - Rosa - que sempre estiveram ao meu lado, me apoiando e motivando a alcançar vôos mais altos. Sem eles, com certeza, nada disso seria possível.

Um agradecimento à SAP Walldorf, em especial meu gerente Arne Schwartz, meu tutor Daniel Ritter, e também Norman May e Thomas Legler que contribuíram imensamente com suas opiniões acerca do projeto desenvolvido. Estendo ainda este agradecimento a todos os estagiários, alunos de PhD e demais funcionários com quem pude conviver tão bem durante o desenvolvimento deste projeto. Meu muito obrigado SAP por ter me permitido crescer tanto profissionalmente durante os meses que pude estagiar.

Não posso deixar de agradecer ao governo francês por toda a ajuda que puderam dar a um estrangeiro em seu país durante uma pandemia para que tornasse não só este projeto possível, mas também a dupla diplomação. Assim, agradeço também à ENSIMAG, em especial à secretária de relações internacionais da escola Elena Leibowitch que incansavelmente atende a todos os alunos internacionais com simpatia e receptividade notáveis.

À professora Renata de Matos Galante, meu imenso muito obrigado pela dedicação ao longo deste último semestre, orientando-me com um acompanhamento fundamental a permitir o bom desenvolvimento desta monografia.

Não posso deixar de agradecer nominalmente a mais alguns professores desta grande jornada acadêmica. Aos meus orientadores de Iniciação Científica: Renato Perez Ribas e Antonio Carlos Beck Filho, obrigado por todo o conhecimento que puderam me passar ao longo de suas orientações. Com certeza esta monografia foi escrita também utilizando muito do aprendizado e conhecimento agregado com os senhores. Ao professor Cláudio Fernando Resin Geyer meu muito obrigado por todo o esforço desempenhado na coordenação do BRAFITEC. E também ao professor Raul Fernando Weber (*in memoriam*), que por meio de suas palavras e aulas sensibilizava a todos os seus alunos com a empolgação pela computação. Em nome deles estendo esse agradecimento a todos os professores que participaram da minha formação, cada um a seu modo me levou a chegar onde estou hoje e estabeleceu um alicerce para alcançar muitos feitos pela frente. “A teacher affects eternity; [they] can never tell where [their] influence stops.” (ADAMS, 1907).

RESUMO

A presente pesquisa pretende analisar o comportamento de Learned Indexes, tendo como ponto de partida o estágio realizado na SAP SE em Walldorf, Alemanha, no período de abril de 2022 a setembro de 2022. Pesquisas recentes sobre Learned Indexes em Sistema de Gerenciamento de Banco de Dados tem demonstrado melhorias significativas de desempenho em relação a índices tradicionais, como B-trees, porém ainda há muito a ser explorado nessa área. Atualmente, há uma carência de estudos sobre dados dinâmicos, principalmente aqueles envolvendo conjuntos de dados reais. Dado o potencial de ganhos significativos de desempenho quando implementados em Sistema de Gerenciamento de Banco de Dados em comparação com os índices tradicionais, há um grande interesse em conduzir um estudo mais aprofundado de Learned Indexes. Assim, o objetivo deste projeto é fazer benchmark de três casos, inicialmente criados artificialmente para que possamos explorar algumas dificuldades do modelo de aprendizado profundo e analisar como o banco de dados se comporta. Na fase inicial deste projeto, estabelecemos uma linha de base reproduzindo resultados de pesquisas anteriores. Observamos que nossos resultados se assemelham bastante ao comportamento relatado nesses artigos. No entanto, ao trabalhar com conjuntos de dados criados artificialmente, notamos overfitting nos modelos utilizados e seu impacto na criação e inserção do modelo.

Palavras-chave: Learned index. arquitetura de banco de dados. .

Learned Indexes with Updates

ABSTRACT

The present research intends to analyze the performance of Learned Indexes, having as starting point the internship held at SAP SE in Walldorf, Germany, between April,2022 and September,2022. While recent research on Learned Indexes within Database Management Systems has demonstrated significant performance improvements over traditional indexes such as B-trees, there is still much to be explored in this area. Currently, there is a lack of studies on dynamic data, particularly those involving real-world datasets. Given the potential for significant gains in performance when implemented in Database Management Systems as compared to traditional indexes, there is a great deal of interest in conducting a comprehensive study of Learned Indexes. So the objective of this project is to benchmark several cases, initially artificially created one so we can explore some deep-learning model difficulties and analyse how the database behaves. In the initial phase of this project, we established a baseline by reproducing results from previous research. We observed that our results closely resembled the behavior reported in those papers. However, when working with artificially created datasets, we noticed some overfitting in the model and its impact on the model's creation and insertion.

Keywords: Learned Index, Database Architecture, .

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS

ALEX Updatable Adaptive Learned Index.

CIT Cumulative Insertion Throughput.

CLT Cumulative Lookup Throughput.

COT Cumulative Operation Throughput.

Database Management Systems (DBMS) are softwares systems used to store, retrieve and run queries on data..

LI Learned Indexes.

LIPP Updatable Learned Index with Precise Positions.

Lognormal Dataset composed by values generated according to a lognormal distribution with $= 0$ and $\sigma = 2$ multiplied by 10^9 and rounded down to the nearest integer.

Longitudes and Latitudes (Longlat) Dataset composed by compound keys that combine longitudes and latitudes from Open Street Maps (OpenStreetMap contributors,).

Longitudes (Long) Dataset composed by longitudes of locations around the world from Open Street Map (OpenStreetMap contributors,).

OSS Open-source software.

PGM-index Piecewise Geometric Model index.

PLA Piecewise Linear Approximation.

RMI Recursive Model Index.

SGBD Sistema de Gerenciamento de Banco de Dados.

YCSB Dataset composed by values representing user IDs generated according to the YCSB Benchmark(COOPER et al., 2010).

SUMÁRIO

1 INTRODUCTION

The use of several algorithms implementing index structures for applications where the need for data access is critical already demonstrate the better efficiency of this type of algorithm over others. Recent experiments and results have shown that Learned Indexes (LI) are a great option to implement Database Management Systems (DBMS) mainly as an evolution from B-trees. In (KRASKA et al., 2018), in which the concept of Learned Indexes (LI) is presented, the authors interpret traditional index structures as models and because of that they assume that these index approaches can then be replaced by other types of models, as is the case with LI that Tim Kraska proposes a deep-learning model.

In the first implementations of such indexes one of the challenges was to implement it with updates, but good results were obtained in recent LI systems. The three systems that we are going to explore more in this project are Piecewise Geometric Model index (PGM-index)(FERRAGINA; VINCIGUERRA, 2020), Updatable Adaptive Learned Index (ALEX)(DING et al., 2020a) and Updatable Learned Index with Precise Positions (LIPP)(WU et al., 2021). However, even with these good results presented, which generates high expectations, there are still challenges to be overcome so we can implement in DBMSs to be sure about the performance gain. Some of these challenges are a lack of real world datasets experiments, usage of different data types, concurrent implementations and a good study over how it behaves under overfitting, as also found in a recent work (WONGKHAM et al., 2022), and caching.

In this project we want to study and analyse some of the behaviors we can expect from a LI system by benchmarking it using first chosen datasets used by other related works, later we use the artificially created datasets with the objective to understand the behaviors over well known data and for the last lasts experiments some real world datasets.

So it can be achieved I started this project making a tracking over the different systems already implemented, that's when it was decided to use the 3 LI systems, the next step was to reproduce the different experiments from these systems. And then finally run our own experiments always focusing on systems that support dynamic data, that's Open-source software (OSS) and that supports relational data model. These characteristics are important for SAP so it's possible to analyse how it works and develops so it's possible to compare to actual products from the company and also decide if it's better to implement

these tools using Learned Indexes as the base for the DBMS.

1.1 Work Structure

The structure of this work is organized as follows. First we introduce the project with a brief overview and main goals in chapter 2 . Then we present the state of the art in chapter 3 where we also present the LIs we use in this project. The methodology and the setup we used for all experiments we present in this document is in chapter 4. Then in chapter 5 we present and discuss the reproductions and the new experiments we did. We discuss related work in chapter 6, conclude and present the future works in chapter 7. In Appendix A an extended abstract from this monograph is presented in Portuguese.

2 PROJECT OVERVIEW

The first part of the project was to track different DBMS that already implemented Learned Indexes. However, some prerequisites were placed as mandatory within this search. The systems I was looking for were LIs that should be Open-source software (OSS) and that supported relational data model. In Table 2.1 the most promising systems were summarized. At this point of the project different data types and concurrency is not so important, but as it should be important in the future we already get this information from the different approaches. So by now all the systems used were single threaded and only accepted number formats – i.e. signed and unsigned integers, double.

Tabela 2.1 – Learned indexes on dynamic data overview (select: OSS, Relational)

System	OSS	Data		Concurrency
		Data Model	Data Types	
PGM (FERRAGINA; VINCIGUERRA, 2020)	👍	Relational	👎	👎
ALEX (DING et al., 2020a)	👍	Relational	👎	👎
LIPP (WU et al., 2021)	👍	Relational	👎	👎
XIndex(TANG et al., 2020)	👍	Relational	👎	👍
SIndex(WANG et al., 2020)	👍	Relational	👍	👍
Tsunami(DING et al., 2020b)	👎	Multi-Dimensional	👍	👎

👍Fully meets the requirement
👍Partially meets the requirement
👎Doesn't meet the requirements but list it as possible future work
👎Doesn't meet any of the requirement

The decision to utilize PGM-index, ALEX, and LIPP was based on their dynamic nature, which allows for updates; their relational structure, which is better suited for the company to adapt to their products; and their open-source status, which ensures legal use. Additionally, they were chosen based on their availability at the time of data collection. The remaining data collected and summarized in Table table 2.1 should be considered for future works.

The list of the settings used for the experiments were also summarized and presented in Table 2.2, and the different datasets used for these experiments in Table 2.3, where the 4 first datasets -Long, Longlat, Lognormal and YCSB- are benchmarks used by ALEX(DING et al., 2020a) and LIPP(WU et al., 2021), and the last three – Web logs, Longitudes and IoT – were used by PGM.

Tabela 2.2 – Experiments on dynamic data

Experiment	Operations	Initial index size	Description	System
Read-only	100% lookup	100M random keys	..	LIPP
Read-heavy	33% insert, 67% lookup	empty	..	LIPP
Write-heavy	67% insert, 33% lookup	empty	..	LIPP
Write-only	100% insert	empty	..	LIPP
Read-Only	100% read	100M	*	ALEX
Read-heavy	95% read, 5% inserts	100M	*	ALEX
Write-heavy	50% read, 50% inserts	100M	*	ALEX
Short range query	95% reads, 5% inserts	100M	**	ALEX
Write-only	100% inserts	100M	..	ALEX

* A read consists in a lookup operation for a single key
** A read consists of a key lookup followed by a scan of the subsequent keys

Tabela 2.3 – Datasets on dynamic data

Data	Num keys	Key data type	Payload size	Total size	Characteristics
longitudes	1B	double	8B	16GB	-
longlat	200M	double	8B	3.2GB	-
lognormal	190M	64-bit int	8B	3.04GB	-
YCSB	200M	unsigned 64-bit int	80B	17.6GB	-
Web logs	715M	timestamp	-	-	web server requests
IoT	26M	timestamp	-	-	IoT sensors*

- Unknown / no entry

2.1 Project goals

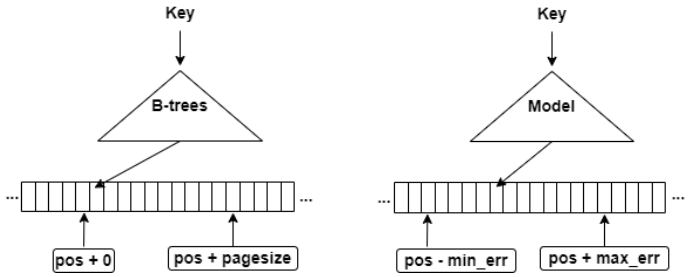
The main goals in this project is to make a study and an analysis over Learned Indexes and its impacts over a DBMS so we can determine if it would be good to implement in SAP products because of the performance gain.

To achieve this goal, different steps and smaller goals must be achieved. The first one was to find LI systems that's already implemented that fulfilled our prerequisites. After that we need to reproduce it to have a basis for comparison with new experiments. From these reproductions and studies over the other projects it's possible then to make an analysis over weak spots and then in which parts we can make improvements. In this moment we already notice some needs to make artificial datasets to test the behaviors over the LI approaches we are testing.

3 STATE OF ART - LEARNED INDEX

Learned Indexes (LI) concept came up in (KRASKA et al., 2018) and the main principle is to understand indices as models. The Figure 3.1 illustrates the main thinking why it's possible to make this replacement. A B-Tree maps from a given lookup key to a range of positions inside an array and the same mapping can be done with a machine learning model that can guarantee a range - between min_error and max_error - in the array since it's sorted and fixed size types.

Figura 3.1 – B-Trees as Models adapted from (KRASKA et al., 2018)



(a) B-Tree Index (b) Learned Index

This first paper on the subject didn't have the purpose to completely replace the traditional index structures for learned index. But it argue that continuous functions can be used to make data structures and algorithms more efficient and from this they conclude that LI has a potential power to be better.

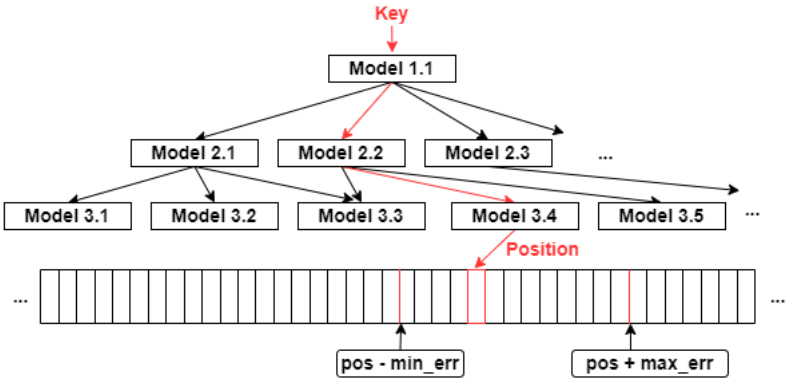
Kraska et al. (KRASKA et al., 2018) also proposed the Recursive Model Index (RMI) that is a recursive regression model built as a hierarchy of models shown in Figure 3.2. The main principle is that in the current stage model the key serves as an input that will then be used to define which model of the next stage will be chosen until the final stage position is predicted. The way it works is as each model would be competent for a certain range over the array - for example the Model 1.1 in the first stage in Figure 3.2 is responsible for all the key space - so the deeper we are in the hierarchy lower is this range and as a consequence we have a better prediction and a lower error.

This staged model is used not only by Kraska, but it's replicated for many of the different systems as all the 3 approaches we chose to replicate: PGM-index, ALEX and LIPP. But they implemented new ways to make this last decision. It will be explained better later on this report, but to give one example ALEX chose not to have a key space and implemented data nodes with gapped arrays.

However, (KRASKA et al., 2018) has its limitations. The implementation focus

only on read-only workloads and designed the RMI statically as a consequence it doesn't support updates and dynamic data.

Figura 3.2 – Staged models adapted from (KRASKA et al., 2018)



With this information from (KRASKA et al., 2018) as the basis for other systems that will be shortly presented. On each of the systems throughout the next sections, the main characteristics and the differences presented in the works will be highlighted in order to show the evolution of the indexes and also to present what will bring the main differences in performance between each of them.

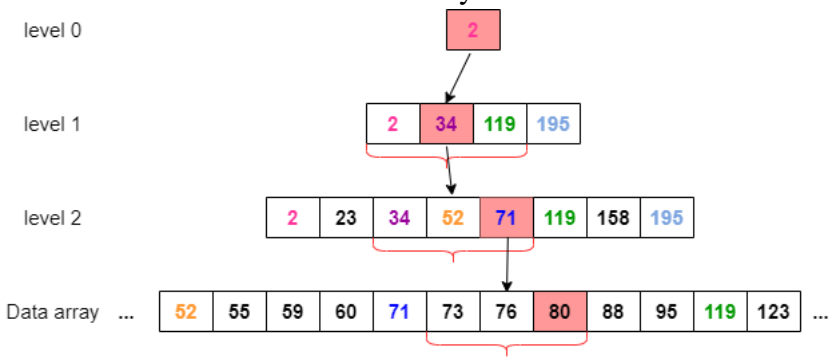
3.1 Piecewise Geometric Model index (PGM-index)

PGM-index was presented in (FERRAGINA; VINCIGUERRA, 2020) where they designed a fully dynamic LI system. They claim to be better than original RMI (KRASKA et al., 2018) in query time and space occupancy and also up to 71% better in query and update time than B+ Trees for dynamic workloads while it can also reduce by up to 4 orders of magnitude the space occupancy.

One of the changes PGM implements is a fixed integer parameter that will control the size of the region that the system will access, as Figure 3.3 shows. These improvements are possible because PGM-index supports distribution-awareness, multi-criteria adaptability, auto-tuned to any given space or latency requirements and compression. This last one is very important for the memory efficiency.

For that we have a Piecewise Linear Approximation (PLA) model that is a mapping between the keys and their approximate position in the data array. The correct position can be at most ε away from the predicted one.

Figura 3.3 – Staged Model adapted from PGM-index(FERRAGINA; VINCIGUERRA, 2020) with $\varepsilon = 1$. The fixed interval is shown by the red brackets

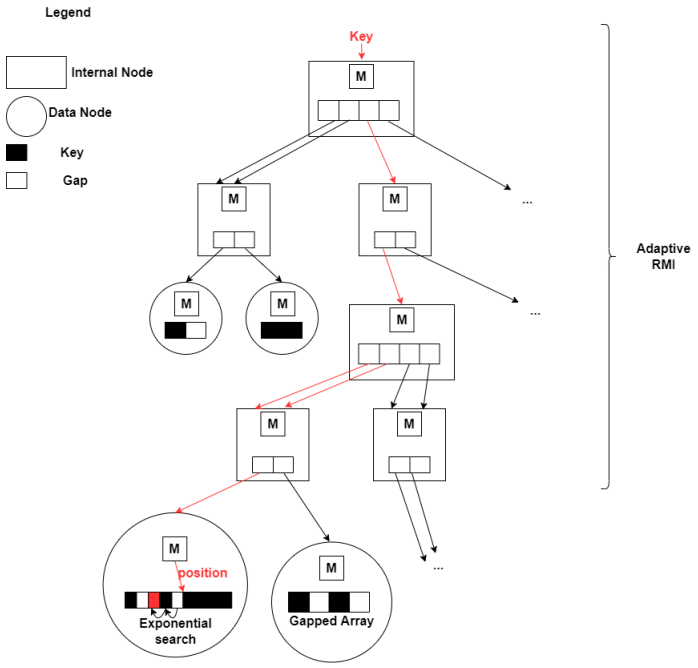


3.2 Updatable Adaptive Learned Index (ALEX)

Different from PGM, that makes improvements but maintained most of how staged models were presented in (KRASKA et al., 2018), ALEX(DING et al., 2020a) introduced a new design for it.

Instead of having a big array at the end, ALEX presents different types of nodes as we can see in the staged model shown in Figure 3.4. The internal nodes, despite appearing similar to those of other systems, will be slightly different as they aim to find a flexible way to partition the key space. And the data nodes will be where the data will be stored in gapped arrays. This way, instead of the last node pointing to a region in a large array that holds all the data, the data will already be dispersed through the different data nodes.

Figura 3.4 – ALEX Staged models adapted from (DING et al., 2020a)



The authors argue for the benefits of using gapped arrays, because in case of a position collision for a given key, instead of already having to divide the nodes - so that more memory is available for the new data - the system will use the gaps, i.e. empty positions, available near the collision for data storage. This generates gain because splitting the nodes generates a drop in performance. ALEX tries again to avoid this by implementing a node expansion mechanism that will create new gaps and with that will decrease the node density again. But inevitably from time to time division will take place. When this division happens, the model is also retrained for the new nodes with the data they will receive.

With all these advances ALEX claims to be up to 2.2 time better on performance with up to 15 times smaller index size than (KRASKA et al., 2018). But thanks to gapped arrays ALEX presents one of the points that has to improve, because inside the data node it performs an exponential search until it finds the correct position. This is one of the main changes presented by LIPP.

3.3 Updatable Learned Index with Precise Positions (LIPP)

LIPP is introduced in (WU et al., 2021). And it presents considerable new modifications in how it implements the staged model. And in place of ALEX's worst case where you would have to do the exponential search all node-wide, LIPP bounds the lookup cost to the tree height. LIPP implements a system such that it precisely maps the key to a position. If too many conflicts start to happen with many keys being taken to the same position, then a new node is created to store these keys.

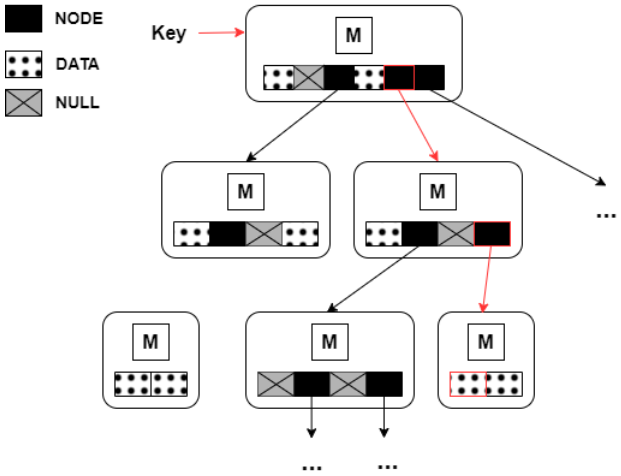
The design presented by LIPP, as shown in Figure 3.5, therefore has an even greater distribution of data than ALEX already did. Each node as well as the rest will have its model, but what changes will be the array of entries, as it can contain 3 types of entries:

- NULL: empty position, similar to gaps implemented by ALEX
- DATA: position that stores the pair <key,payload>
- NODE: A pointer to a child node at the next level.

This is how conflicts are resolved. If the system tries to insert a tuple <key,payload> pair in a data position, a new node is created to store both data and the position that previously stored data starts to point to the newly created child node.

LIPP also presents some strategies to fit the models in order to keep the tree height

Figura 3.5 – LIPP Staged models adapted from (WU et al., 2021)



bounded. As the worst case of LIPP is given by the depth of the tree, this is a very important mechanism to maintain performance. According to the authors, then the criteria for the adjustment decision are that the number of elements inserted in the subtree is at least β times the number of elements that this subtree contained in the previous adjustment. The second criteria is that the rate of conflicts and insertions in a subtree exceeds a α threshold. Both α and β are implementation-defined and can be changed. By default the paper displays $\alpha = 0.1$ and $\beta = 2$. If the systems meet any of these criteria, the system will run the adjustment algorithm so that the tree gets a lower height.

3.4 GRE

GRE is a framework suite for traditional indexes and Learned Indexes , which is presented in (WONGKHAM et al., 2022), to measure throughput and latency. The project aims to evaluate the practicality of using learned indexes in real DBMS and makes two major contributions to the community.

The first contribution, which I consider the most significant in the paper, is a quantitative measurement that uses the Piecewise Linear Approximation (PLA) to estimate the hardness of a dataset for a specific LI. This approach enables a better understanding of the usage of different indexes under varying scenarios.

The second contribution is the open-source benchmarking suite available at (WU et al., 2022). This tool helps new studies over LIs making easier to evaluate learned over traditional indexes. It also makes available scripts to visualize the experiments done.

Although the framework was available, it was not utilized in this monograph. This

was due to the fact that at the time of the conducted experiments, GRE had not yet been made available. The publication of this framework occurred long after the start of the experiments that are to be presented in this paper.

4 METHODOLOGY

This section will present our methodology and setup to run all the experiments.

4.1 Execution Methodology

As a methodology for the experiments we followed the methodology presented in (RITTER et al., 2016) in which the experiment is divided in forks and each of the forks will run warmups and a determined number of iterations from the required program as it's shown in Figure 4.1(a) how it's supposed to work. For that the environment should be completely cleaned for each fork in that way we can minimize problems as caching and also that can make us have a more confident result, because we run several forks for each experiment as many as needed so we have high confident interval - 95 or 99%.

All the experiments that has been done and presented in this document passed through the same method. We run 3 forks / runs for each experiment - that's the combination between the different settings and the different datasets - each of these forks consisted of 4 warmup cycles and then 5 iterations of the program, one fork cycle can be seen in Figure 4.1(b) starting from building the docker container. Each iteration had as output a csv file that contains the performance data that we used to get the results we will present in the next sections, after each fork we copy all output csv files from the container to the server.

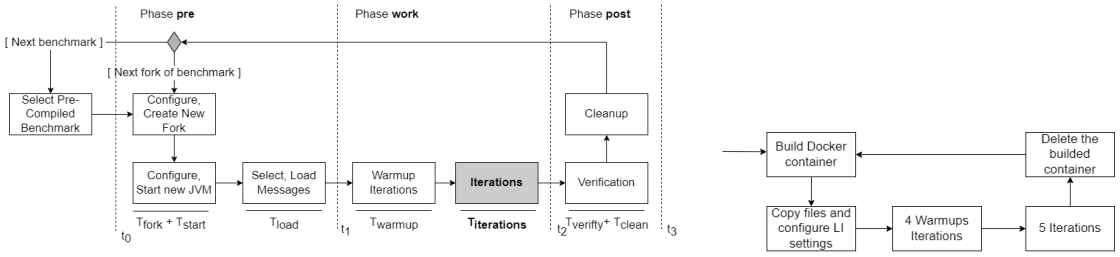
For the results that are going to be presented in the next sections are the mean of the cumulative throughput from 3 forks. For each fork we completely clean the containers rebuilding from scratch each time this gives me a better way to calculate the confidence interval and also prevents any caching that can happen from execution to execution.

Thus, the main metric used in this work was the Cumulative Operation Throughput (COT) However, to better analyze the results, Cumulative Lookup Throughput (CLT) and Cumulative Insertion Throughput (CIT) were also utilized. These metrics allowed for a better understanding of the accumulated behavior in the and to determine which metric had a greater weight within that specific experiment. These metrics can be calculated by the formulas in Table table 4.1

Tabela 4.1 – Formulas to calculate the different Throughputs

Metric	Formula
COT	$\frac{\text{Lookup Operations} + \text{Insertion Operations}}{\text{Lookup Time} + \text{Insertion Time}}$
CLT	$\frac{\text{Lookup Operations}}{\text{Lookup Time}}$
CIT	$\frac{\text{Insertion Operations}}{\text{Insertion Time}}$

Figura 4.1 – Execution phases methodology



(a) EIPBench execution phases Adapted from (RITTER et al., 2016)

(b) One fork execution

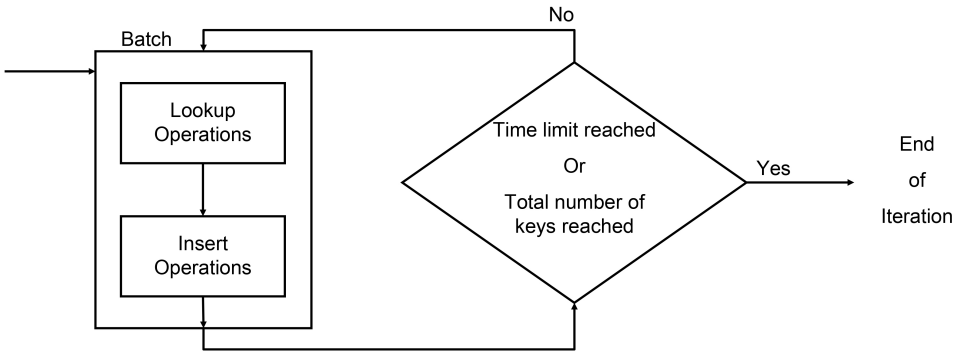
4.2 Experiment's Setup

All the experiments were executed in docker containers in a dedicated server that has an Intel(R) Xeon(R) Silver 4116 CPU @ 2.10GHz. All codes were implemented using C++ and to run all the experiments following the methodology presented before all the commands were run using shell scripts.

For all DBMS each iteration, that's execution of the program, consists of a bulkload of a given number of keys and after that the program enters in a loop up to a time limit or the total number of keys - also given to the program - is reached. Each of the loop iterations it's called a batch as we can see in Figure 4.2 . We also define the number of operations each batch will run and the percentage of these operations that's going to be insertions and by consequence the rest of it is going to be lookup operations.

The base numbers we used in the experiments we are going to present the results are a starting bulkload of 10 million keys, a total number of 20 million keys - that is the limit defined so the loop stops if met - and 1 million operations per batch. As it's possible to see in Table 2.2 the main parameter that changes the different workloads are the percentage of insertion and lookup operations. So for example when we define 5% insertion operations each batch will consist of 50 thousand insertion operation and 950 thousand lookup operations. Any experiment that does not follow these base numbers will be explicitly explained in the specific experiment in what the changes were made.

Figura 4.2 – Batch algorithm extended and adapted from (DING et al., 2020a)



5 EXPERIMENTS

This chapter will detail the experiments conducted and also provide a brief analysis of the results obtained from those experiments.

5.1 Learned Index Replications

We start replicating the 3 Learned Indexes (LI) that we chose. We used ALEX benchmark code as a platform to reproduce the others implementing PGM and LIPP libraries in it. All of them in separated codes. So we have the same measurement pattern for all the three of them. The objective here is to compare our execution with the the tests made and presented by the original papers.

For better visualization reasons in the diagrams presented below as results of the reproductions made, we changed the names of the datasets by acronyms presented in Table 5.1

Tabela 5.1 – Datasets acronyms

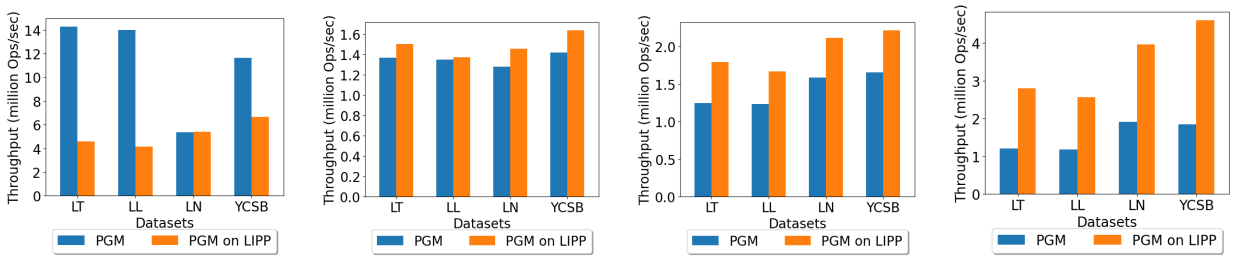
Longitudes	LT
Longlat	LL
Lognormal	LN

So for the replication we followed exactly the way presented in Section 4. As output we got csv files that contained the Cumulative Operation Throughput (COT) – i.e. the throughput that we got from lookup and insertion operations – for each one of the batches. So we make the mean of these throughput data and that's the value used to make the diagrams. We also analysed the standard variation of the data we are using so we can be sure that the mean was a reliable value to represent the experiment result. In all cases of experiments, the standard deviation found in the data is orders of magnitude lower than the mean value. In addition, no large outliers were found within the data collected for calculating the average Cumulative Operations Throughput.

As it is shown in Figure 5.1 and Figure 5.3 the compared values came from LIPP's paper reproduction data for those 2 replications. PGM-index originally used other datasets and also other metrics for its evaluation. So for consistency and to have a baseline to compare we take the results that LIPP presented (WU et al., 2021) and used that as data in Figure 5.1.

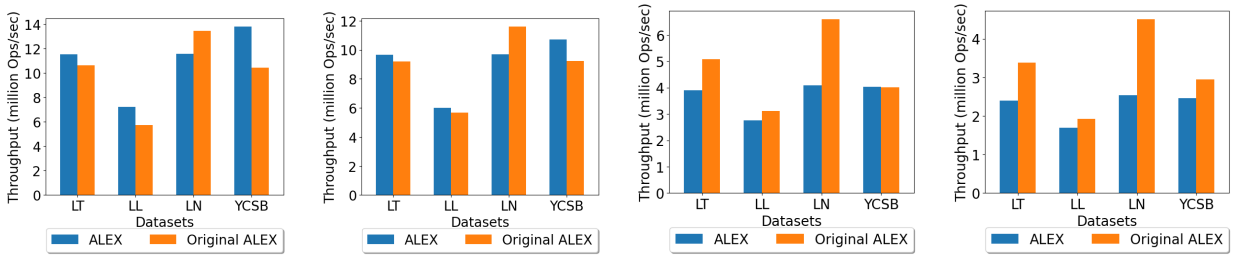
Thus, with result diagrams we can finally analyse the data we got. ALEX replica-

Figura 5.1 – PGM replication data compared with LIPP reproduction



0% insertion 33% insertion 66% insertion 100% insertion

Figura 5.2 – ALEX replication



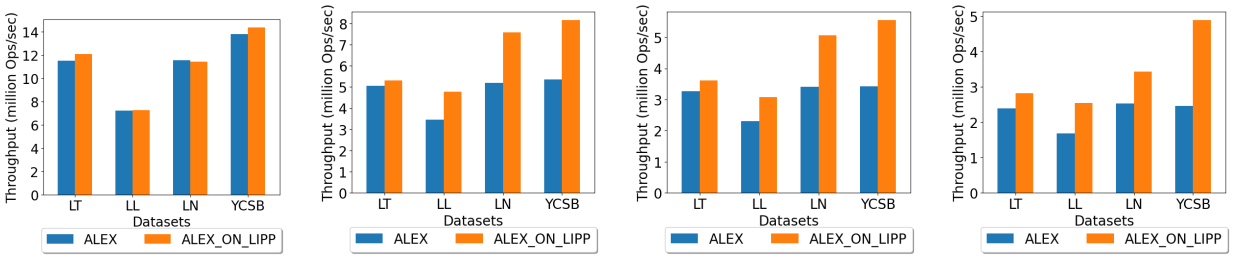
0% insertion 5% insertion 50% insertion 100% insertion

tion, Figure 5.2, did not reach the same numerical value in the throughput measurements, but we noticed that the execution for the Longitudes (Long) and Longlat datasets show the same tendency on both configurations in Figure 5.2 and Figure 5.3. However, the findings revealed that the Lognormal and YCSB datasets had different behaviors than in the ALEX paper, but it's close to the behavior LIPP presented in it's paper as we can see on Figure 5.3. Perhaps this is due to some execution difference on the part of the ALEX development team. Although among all 3 systems, this is the one that contains the best documentation and description of how to perform benchmarks on the system. Which made it the easiest to reproduce.

For LIPP replication we can make a similar analysis that as we can see in Figure 5.4 we have similar behavior. For the PGM-index reproduction that the results found for 0% insertion is the most outlier one and need further investigation with more experiments, because as we can see in the Figure 5.1 it got very different throughput than LIPP reproduced.

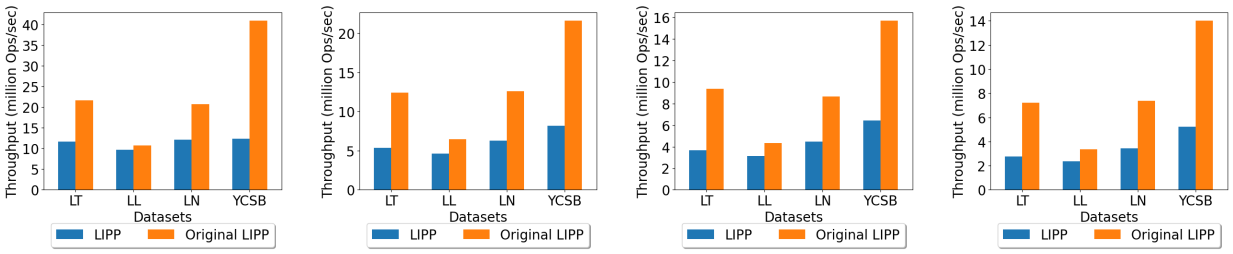
As we can see from Figure 5.5, the data collected from 0% insertion shows an anomalous behavior compared to the rest of the results collected. This indicates a very high lookup throughput, which makes the COT result so far from the LIPP reproduction, as shown in Figure 5.1. Therefore, further investigation is needed to understand why this behavior occurs.

Figura 5.3 – ALEX replication data compared with LIPP reproduction



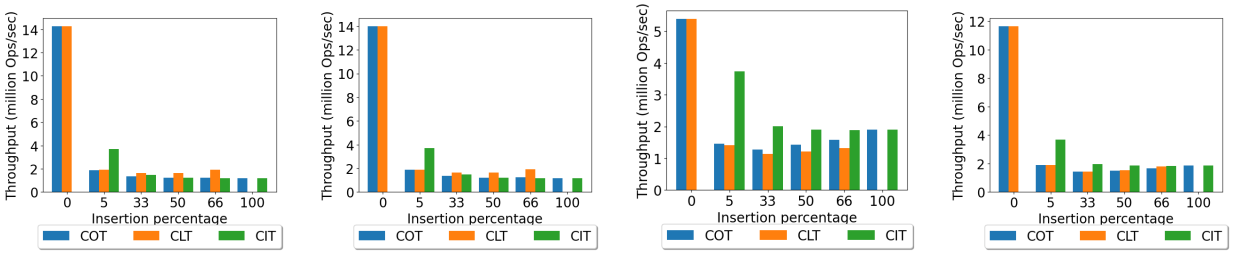
0% insertion | 33% insertion | 66% insertion | 100% insertion

Figura 5.4 – LIPP replication



0% insertion | 33% insertion | 66% insertion | 100% insertion

Figura 5.5 – 0% insertion on PGM replication



Long | Longlat | Lognormal | YCSB

5.2 New Experiments

After we ran all the replications and got all that data as a baseline for the next steps. Differently then (WONGKHAM et al., 2022) we started our experiments with artificially created datasets. They preferred to start directly with real world datasets including SOSD's (KIPF et al., 2019) datasets, but we would like to have the results of this data created by us. The objective in this part was to see the behavior of the different LIs on data that we knew exactly what it was and how it was getting inserted in the database. One of the tests we wanted to see is how the DBMS behaves when we force a overfitting in the models used by it. After that we started real world datasets tests. All these tests have as objective to simulate different characteristics that may occur in customer data. The datasets used by the time this report was done is summarized in Table 5.2.

Tabela 5.2 – New Datasets used in experiments

Data	Number of keys	Key data type	Payload size	Total size	Characteristics
minMax	200M	int	4B	1.99GB	Artificially created by us
density	200M	int	4B	1.84GB	Artificially created by us
random	200M	int	4B	2.13GB	Artificially created by us

All artificially created datasets were generated using Python scripts that employed integers as data types and saved the resulting integers in a text file. Each line of the file corresponds to a single number to be used as key in the experiments.. So we created the following datasets:

- **minMax**: Made only with odd numbers. The first part is low odds starting at 1, the number for this low odd integers is enough so we populate the database with them in the bulkload. The second part is made of high odds starting in INT_MAX from C++ that's 2147483647 and going down. This is supposed to provide us data so we can analyse the behavior of the DBMS when the model is started with only low integers, overfitting the models on that way, then started to face high integers in the insertion process
- **density**: The bulkload part is made with compact and low odds as the minMax. And the second part to evaluate the model created we start insertion the even numbers in-between the odd ones. Here the objective is to analyse how the model behaviors when so near and compact data is inserted.
- **random**: a dataset composed by 200 million integer keys. A uniform random algorithm was used to generate it.

With these datasets the objective is to force an overfitting in the models with *min-Max* and then see how the different systems behavior with that. With *density*, there's going to be some overfitting also, but the main test is to see how effective the systems will become with so dense data and by consequence with the need to deal with key conflicts and node splits. *Random* is actually a good baseline dataset, because we can discard that the behave we are getting is not the same as if we got just random keys, and even being random as soon as it's uniformly random integers we can also check how the different systems behave with this uniform distribution.

In Figure 5.6 we can see an extract from minMax and Density datasets. This is to show visually the created data so it can be easier to understand what's being used. These 2 datasets were created with 20 keys long file, the real datasets used in the experiments are 200 million keys long files.

Figura 5.6 – Datasets examples

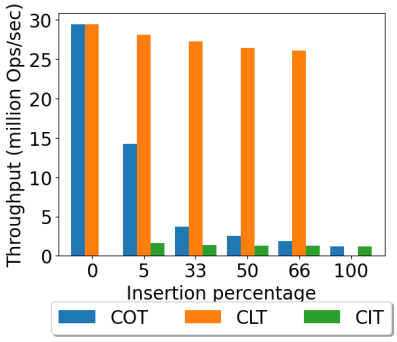
1	1
3	3
5	5
7	7
9	9
11	11
13	13
15	15
17	17
19	19
2147483647	0
2147483645	2
2147483643	4
2147483641	6
2147483639	8
2147483637	10
2147483635	12
2147483633	14
2147483631	16
2147483629	18

minMax dataset density dataset

5.2.1 Results

First, other experiments have been done one of them was an experiment starting the batches without the bulkload. The measured performance varied so much in the initial batches that the standard deviation of this data was very high and clearly the mean was not a reliable data. Therefore, we can conclude that the impact in the initial performance

Figura 5.7 – ALEX running minMax dataset



of filling the database after a few insertions is lessened and from what we could see in the results it ends up stabilizing after some batches. With that we can say that it's important that the database is initialized with data through the initial bulkload. This makes the system better prepared to receive new data.

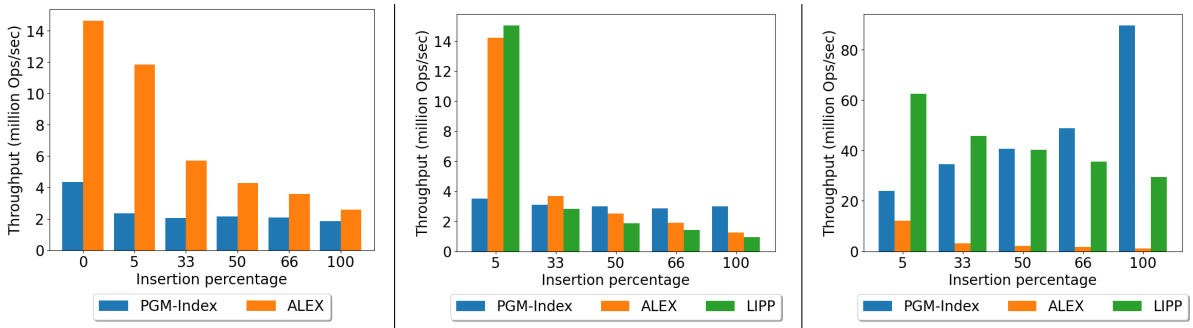
For these new experiments we also notice a very high throughput for 0% insertion when we compare to the others results in the same dataset and just changing the insertion rate, for example running the minMax dataset from 0% to 5% insertion the system that got the least performance decrease was ALEX and already got 51.53% lower throughput. The conclusion we got from this is that the usage of these artificial datasets that try to exploit overfitting, but at the same time simple as they use of a standard interval between 2 consecutive keys, allowed the model to adapt very well to the search input keys.

In the next diagrams the measurement shown is the Cumulative Operation Throughput (COT) as the ones in reproductions. The COT is calculated by the division from the sum of lookup operations with insertion operations by the sum of the time to run each one of them. This is important to fully comprehend Figures 5.7 and 5.9. In those two figures, in addition to the COT, the Cumulative Insertion Throughput (CIT) and Cumulative Lookup Throughput (CLT) are also presented.

In Figure 5.7 it is possible to see that the lookup operations are very fast in the opposite side we have that insertion operations is too slow. Not to be misunderstand here, in this point when we say fast and slow, we are comparing the throughput data we are showing, so by fast we mean high throughput and by slow lower throughput. Part of this is because of overfitting the models and also because having to split the nodes for the new entries makes the LI to lose performance.

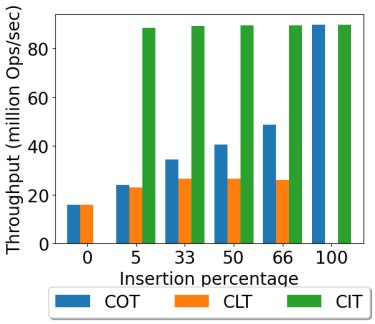
With this last conclusion in mind, in Figure 5.8 we can see a behavior not so positive for most of the results, especially for systems that need regular updates, as the percentage of insertion increases the performance drops considerably. The random data-

Figura 5.8 – Cumulative Operations Throughput results



(a) Random (b) minMax (c) Density

Figura 5.9 – PGM running density dataset



set, despite having losses, still shows a different behavior, probably due to the distribution of the bulkload values that allowed better models. However, the experiment that stands out is PGM-index running the density dataset. As we can see in Figure 5.9, it is the experiment in which, unlike the previous examples, it increases its throughput of operations thanks to the throughput of insertions. What is probably helping this performance is the way PGM-index works using a fixed size – as shown in Figure 3.3 – for the range in which the system accesses and the nature of the dataset for containing such approximate data.

6 RELATED WORK

Learned Indexes work across a wide spectrum of computing areas. There are research works related to LIs in the area of machine learning, database architecture and indexing techniques. However, in this chapter I will highlight the main related works.

B-tree and its variations are the algorithms that have made great strides in recent years. And the base on which (KRASKA et al., 2018) started as a point to be evolved to present Learned Indexes. The systems themselves presented during this report (KRASKA et al., 2018), PGM-index(FERRAGINA; VINCIGUERRA, 2020), ALEX(DING et al., 2020a) and LIPP(WU et al., 2021) are also some of the related works. In Table 6.1, adapted from LIPP(WU et al., 2021), we have a summary of the complexities of the indices presented in comparison with a B+ tree.

As already listed in Table 2.1, XIndex implements a LI that supports multicore data storage. And they claim to handle concurrent writes without dropping the query performance. The authors have implemented two OSS indexes: XIndex-R which implements a concurrent learned range index and XIndex-H that is a concurrent learned hash index (XINDEX,).

SIndex and Tsunami are also important examples of the fast evolution of Learned Indexes (LI), as they have ready implementations for different data models and data types. While they may not meet the data model requirements for our project, they serve as good examples of LI's versatility. SIndex, for instance, supports string data types. Another notable example is LIFOSS (YU et al., 2022), which implements Learned Indexes that support data streaming.

A great highlight is (WONGKHAM et al., 2022), a recent work that was published at the end of this project. They had a similar objective to my project to present a better study on LI with updates and check how the current implementations are doing. And they implemented and made available an OSS tool that serves as a benchmarking suite. The authors also presented a quantitative metric to evaluate the difficulty that a Learned Indexes has to deal with a given dataset. The algorithm proposed to approximate to calculate this metric in (WONGKHAM et al., 2022) is the PLA. The workloads from them are slightly different from the ones used in this project, but they still use the same idea about changing the rate between lookup and insertion operations. One huge difference is that they focus only on real world datasets some of them presented in SOSD(KIPF et al., 2019).

SOSD (KIPF et al., 2019) is a benchmark framework to learned index consists of different number type datasets – 32-bit or 64-bit unsigned integers sorted data – that propose this data with baseline implementations so different LIs can be tested with it. But they are still in progress with data with updates that's important for this project.

Tabela 6.1 – Summary of complexity comparisons adapted from LIPP(WU et al., 2021)

		PGM-index	ALEX	LIPP	B+ Tree
Lookup operations	Complexity	$O(\log^2 N)$	$O(\log N + \log m)$ [1]	$O(\log N)$	$O(\log N)$
Insert operations	Complexity	$O(\log^2 N + \log N)$ [2]	$O(\log^2 N + \log m)$	$O(\log^2 N)$	$O(\log N)$
2*Search range	Leaf	$O(\varepsilon)$ [3]	$O(m)$	$O(1)$	$O(m)$
	Non-leaf	$O(\varepsilon)$ [3]	$O(1)$	$O(1)$	$O(m)$

1 - m is the max number of slots in nodes

2 - Insertion include checking for key existence

3 - ε is the prediction error threshold

7 CONCLUSION

This exploratory project was extremely gratifying and brought great insights into the use of LI as part of a DBMS.

Learned Indexes since they were introduced by (KRASKA et al., 2018) have shown themselves as a new DBMS with a high potential. With the results obtained so far, I believe that they can really have a gain over traditional Index structures. However, they still need more features implemented so that they are really capable of replacing current systems.

Some works are already following the research to support other data types and also to make LI concurrent. And this is a fast evolving area with new proposals in multi-dimensional data(DING et al., 2020b) and different data types as strings (TANG et al., 2020). With the potential that these systems have shown together with the knowledge of performance gain from neural networks running on GPUs or specific hardware, if an implementation succeeds it is very likely that it will far exceed traditional rates and most likely there will be no more doubts about the exchange.

Some of our negative results brought a good knowledge about problems that Learned Indexes can face and also a spectrum of characteristics to be avoided in datasets for this type of system. With the behaviors found in the new datasets we have also noticed a high degree of impact with overfitting and that, therefore, the machine learning difficulties will also be challenges for systems that implement Learned Indexes as an algorithm.

In order to pursue further research in Learned Indexes, it is crucial to test Learned Indexes (LI) using more robust real-world datasets. A more in-depth study is also needed to understand how caching, locality, and data size can influence the behavior of a Database Management System (DBMS) with LI. Currently, various approaches utilize Monotonically Increasing Models and sorting prior to bulkloading. However, an essential next step is to implement a system that does not require data to be sorted.

In addition, future work on Learned Indexes (LI) should focus on implementing concurrency, possibly with a speculative locking protocol. As presented in the report, the current systems perform their implementations and tests using only one core, limiting their scalability. Moreover, it is important to extend LI's capability to handle other data types such as strings.

REFERÊNCIAS

ADAMS, H. **The Education of Henry Adams**. [S.l.]: Houghton Mifflin, 1907.

COOPER, B. F. et al. Benchmarking cloud serving systems with ycsb. In: **Proceedings of the 1st ACM symposium on Cloud computing**. [S.l.: s.n.], 2010. p. 143–154.

DING, J. et al. Alex: an updatable adaptive learned index. In: **Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data**. [S.l.: s.n.], 2020. p. 969–984.

DING, J. et al. Tsunami: A learned multi-dimensional index for correlated data and skewed workloads. **arXiv preprint arXiv:2006.13282**, 2020.

FERRAGINA, P.; VINCIGUERRA, G. The PGM-index: a fully-dynamic compressed learned index with provable worst-case bounds. **PVLDB**, v. 13, n. 8, p. 1162–1175, 2020. ISSN 2150-8097. Disponível em: <https://pgm.di.unipi.it>.

KIPF, A. et al. Sosd: A benchmark for learned indexes. **arXiv preprint arXiv:1911.13014**, 2019.

KRASKA, T. et al. The case for learned index structures. In: **Proceedings of the 2018 international conference on management of data**. [S.l.: s.n.], 2018. p. 489–504.

OpenStreetMap contributors. **OpenStreetMap on AWS**. <https://registry.opendata.aws/osm/>. Last access at March, 24th 2023.

RITTER, D. et al. Benchmarking integration pattern implementations. 2016.

TANG, C. et al. Xindex: a scalable learned index for multicore data storage. In: **Proceedings of the 25th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming**. [s.n.], 2020. p. 308–320. Disponível em: <https://ipads.se.sjtu.edu.cn:1312/opensource/xindex>.

WANG, Y. et al. Sindex: a scalable learned index for string keys. In: **Proceedings of the 11th ACM SIGOPS Asia-Pacific Workshop on Systems**. [s.n.], 2020. p. 17–24. Disponível em: <https://ipads.se.sjtu.edu.cn:1312/opensource/xindex/-/tree/sindex>.

WONGKHAM, C. et al. Are updatable learned indexes ready? **arXiv preprint arXiv:2207.02900**, 2022.

WU, J. et al. Updatable learned index with precise positions. **Proc. VLDB Endow.**, v. 14, n. 8, p. 1276–1288, 2021. Disponível em: <http://www.vldb.org/pvldb/vol14/p1276-wu.pdf>.

WU, J. et al. **GRE**. 2022. <https://github.com/gre4index/GRE>. Last access at March, 24th 2023.

XINDEX. <https://ipads.se.sjtu.edu.cn:1312/opensource/xindex>. Last access at March, 24th 2023.

YU, T. et al. Lifoss: a learned index scheme for streaming scenarios. **World Wide Web**, v. 26, p. 1–18, 02 2022.

APÊNDICE A — RESUMO EXPANDIDO

Este capítulo tem por objetivo apresentar o resumo expandido desta monografia, uma vez que o texto completo está escrito em inglês.

A.1 Introdução

A apresentação dos conceitos de Learned Indexes por Tim Kraska em 2018 (KRASKA et al., 2018) já apresentava um potencial de melhoria na performance em relação ao índices tradicionais. Desde então, foram apresentados diversos outros artigos e implementações cada um trazendo uma melhoria ou uma implementação de um novo recurso. Apesar da evolução nos últimos anos, ainda é uma área com muito espaço de exploração e que pelo potencial envolvido traz muito interesse à comunidade de arquitetura de banco de dados.

Diversos algoritmos e programas implementam estruturas de índices quando a aplicação requer uma alta eficiência de acesso aos dados. Tradicionalmente, em Sistemas de Gerenciamento de Banco de Dados já temos implementados o uso de *B trees* ou *B+ trees*, com isso recentes estudos apontaram uma forma de tornar esse uso de índice ainda mais eficiente, tornando encontrar o valor desse índice mais performático com a utilização de *Learned Indexes* (LI).

A apresentação desses conceitos por Tim Kraska em (KRASKA et al., 2018) fomentou o crescimento de pesquisa nessa área. No artigo, é apresentado a uma interpretação sobre as estruturas tradicionais de índices como uma forma mais genérica pensando em modelos. Pode-se utilizar qualquer modelo, evidentemente buscando melhorias em algum quesito computacional como por exemplo performance, segurança, confiabilidade entre outros. No artigo de Kraska e nos outros Sistema de Gerenciamento de Banco de Dados (SGBD) apresentados neste trabalho todos focaram no quesito de ganho de eficiência com a utilização de modelos de aprendizado profundo.

O contexto de desenvolvimento desse Trabalho de Conclusão se deu no time de arquitetura de base de dados da SAP em Walldorf, aspecto esse importante para algumas das decisões tomadas durante a execução do projeto. Com as primeiras implementações de LI tendo dificuldades de implementar atualizações - sendo que alguns recentes resultados obtiveram sucesso em implementar isso como visto em (FERRAGINA; VINCIGUERRA, 2020), em (DING et al., 2020a) e em (WU et al., 2021) - sendo outros aspectos ainda

pouco explorados ou com poucas implementações efetivas como a concorrência e um conjunto de testes com melhor proximidade a dados de mundo real.

As características procuradas em *Learned Indexes* já implementadas foram que fossem Open-source para que, dessa forma, fosse permitido o uso da implementação; que modelassem dados relacionais, pois isto é o que atenderia aos bancos de dados da empresa e de clientes; e que permitissem atulização em seus dados como o próprio título desta monografia já delimita. Ainda com o objetivo de já pensarmos em projetos futuros a esta monografia, também exploramos outras características como o suporte a outros tpos de dados, além de numéricos, e sobre o suporte a concorrência. Os principais trabalhos encontrados estão sumarizados na Tabela 2.1 no Capítulo 2. Com isso determinado, escolhemos 3 destes índices para reproduzir e testar com conjuntos de dados criados artificialmente para a análise da possibilidade de Learned Indexes serem mais eficientes que índices tradicionais.

A.2 Trabalhos relacionados

Learned Indexes pode ser relacionado a trabalhos de diferentes área da computação a ele associadas. Como aprendizado de máquina, arquitetura de banco de dados e técnicas de indexação. Mantendo o escopo um pouco mais contido, focarei nos trabalhos desenvolvidos como Learned Indexes.

B-trees e suas variações, única exceção de trabalho relacionado que não é um Learned Index, são a base no qual (KRASKA et al., 2018) se utilizou não só como linha de base de resultados, mas também de onde partiu a substituição deste algoritmo por um modelo de aprendizado profundo. Este índice, assim como PGM-index, ALEX e LIPP são todos relacionados a esta monografia e são melhor explorados no capítulo 3 que versa sobre o estado da arte.

Temos também um grupo considerável de LIs que ficaram fora do escopo desta monografia por conta de não estarem dentro das características desejadas. Alguns exemplos são o XIndex através de sua concorrência, Tsunami com um modelo de dados diferente do que procurávamos e um mais recente o LIFOSS (YU et al., 2022) que apresenta uma implementação de Learned Index para um streaming de dados.

Há ainda o framework GRE (WONGKHAM et al., 2022) que é um trabalho com um estudo semelhante ao apresentado nesta monografia. Neste projeto, além de apresentarem o framework para a execução de testes em diferentes índices, eles também trazem

uma forma de calcular uma métrica para medir a dificuldade que um dado conjunto de dados tem para ser usado com um LI. Este último aspecto é uma contribuição muito positiva para a comunidade, uma vez que um dos pontos ainda a serem evoluidos é o uso de conjuntos de dados reais. Com este cálculo pode-se ter uma primeira métrica na análise da escolha de implementar ou não um LI como parte do SGBD a ser desenvolvido.

Ainda a adicionar o SOSD (KIPF et al., 2019) que é um benchmark para *Learned Indexes* composto por diversos conjuntos de dados. Os conjuntos de dados disponibilizados são compostos por dados numéricos - inteiros sem sinal de 21 ou 64 bits - ordenados. Porém, é um *framework* em desenvolvimento, ainda não tendo uma preparação muito grande para testes com dados dinâmicos além de ainda estarem buscando em seus conjuntos de dados uma forma de simular conjuntos de dados reais.

A.3 Metodologia

Iniciamos o projeto definindo o escopo de pesquisa e selecionando dentre os diversos índices encontrados quais os Learned Indexes (LI) que iríamos replicar. Com isto definido, começamos os experimentos com a metodologia que será a seguir apresentada.

A metodologia de experimentos utilizada neste projeto foi a apresentada em (RITTER et al., 2016). Esta metodologia consiste em realizar os experimentos repetidas vezes em ciclos de execução compostos pelas fases: pré-experimento, trabalho e pós-experimento. Para cada um destes ciclos o ambiente de experimentação deve estar completamente limpo com o objetivo de minimizar efeitos de caching e qualquer aproveitamento de cálculos de experimentos passados. O objetivo de realizar repetidas vezes é o de aumentar o intervalo de confiança das métricas e dados obtidos.

Durante a fase de pré-experimento é realizada a configuração do ambiente. No caso desse projeto, a fase de pré-experimento consiste em criar o container Docker, copiar e compilar o LI a ser testado e copiar os arquivos de conjunto de dados.

Em seguida, durante a fase de trabalho são realizadas 4 iterações de warmup seguidas de 5 iterações em que os dados serão capturados. O objetivo de se realizar warmup antes da coleta de dados é para que se evite as variações de um ambiente completamente novo. No caso deste projeto, os experimentos passaram por ciclos compostos por 4 iterações de warmups e 5 iterações para coleta de dados.

Para encerrar o ciclo ocorrerá, então, a fase de pós-experimento que é o momento em que as verificações são feitas e em que a limpeza do ambiente ocorre. Esta fase

se resumiu a copiar os arquivos com as métricas obtidas durante as iterações e então a remoção do container. Com isso, a cada ciclo de experimentos, temos um container completamente novo, e portanto garantimos o início com o ambiente limpo.

A coleta de dados se deu em 3 destes ciclos de execução para cada combinação de experimento que se dá através de LI, configuração e conjunto de dados.

A.4 Resultados

Os resultados obtidos se deram em 2 fases. Uma primeira de replicação, momento em que testamos os resultados obtidos dentro do ambiente configurado para este projeto e comparamos com os resultados obtidos e apresentados pelos trabalhos originais. Num segundo momento, foram realizados testes utilizando conjuntos de dados criados para testarmos alguns comportamentos da modelagem dos índices. Vale ressaltar que a principal métrica utilizada para avaliação destes experimentos foi a vazão do índice medida em milhões de operações por segundo.

Os LIs utilizados como base para a replicação e experimentos foram PGM-index (FERRAGINA; VINCIGUERRA, 2020), ALEX (DING et al., 2020a) e LIPP (WU et al., 2021) . Durante a fase de replicação, apresentada no capítulo 5, foram executados os mesmos conjuntos de dados dos projetos originais: Long, Longlat, Lognomal e YCSB. Com a configuração de um carregamento inicial de 10 milhões de chaves, definindo um número total de 20 milhões para os experimentos e um total de 1 milhão de operações por batch de operações. A configuração que variou dentro da combinação apresentada no item anterior é da proporção entre inserções e operações de busca. As configurações utilizadas pelos trabalhos originais estão sumarizadas na tabela 2.2 e estão explicitadas nos diferentes gráficos do capítulo 5.1.

Depreende-se dos resultados obtidos nessa fase que nosso ambiente em diversos casos não alcançou os números absolutos dos trabalhos originais. Porém, em diversos casos o que podemos perceber é uma tendência de comportamento similar.

Um dos grandes pontos fora da curva que se destacou durante a comparação de resultados foram os resultados obtidos na reprodução de ALEX. Quando comparado ao trabalho original, dois dos conjuntos de dados pareciam invertidos entre si em comportamento, mas que ao serem comparados com a reprodução feita por LIPP do LI apresentado por ALEX estes tinham um comportamento muito similar ao obtido por nós.

A segunda fase de experimentos se deu com testes utilizando alguns conjuntos de

dados artificiais. O objetivo foi explorar algumas fragilidades esperadas de modelos de aprendizado profundo, como overfitting, e também analisar métricas do comportamento dos LIs quando submetidos a estes tipos de conjunto de dados. Nesta fase, alguns outros experimentos também foram realisados com o objetivo de uma melhor compreensão das etapas da execução dos algoritmos. Porém, o destaque se dá nos conjuntos de dados criados para a execução dessa fase com seus resultados e novas percepções que tivemos após a realização dos testes.

Os conjuntos de dados criados foram *minMax*, *density* e *random* apresentados na seção 5.2. Pudemos perceber um resultado diferente por conta do *overfitting* dos modelos que quando submetidos a dados ao mesmo tempo que o diferente comportamento entre os índices também demonstram na prática as características que diferenciam um índice dos outros.

A.5 Conclusão

Desde a apresentação de Learned Indexes por (KRASKA et al., 2018) tem-se tido resultados positivos em relação a modelos clássicos de índice - como B trees - numa implementação de um SGBD. As evoluções que diferentes LIs nos últimos anos vem no direcionamento de também pensarem que há um grande potencial de melhora na performance de SGBDs e buscarem essa comprovação.

Como este foi um projeto bastante exploratório sobre a área, destaca-se uma comunidade bastante ativa e com diversos projetos sendo apresentados com evoluções em diferentes aspectos de LIs.

Ao término de nossos experimentos, concluímos que Learned Indexes tem sim um grande potencial. Porém, as implementações atuais ainda tornam de difícil utilização num SGBD como um produto final. Alguns de nossos resultados negativos e dificuldades durante a execução deste projeto mostram muito bem alguns dos desafios atuais que LIs ainda precisam superar. A complexidade dessas soluções vem da necessidade de implementação de recursos em diferentes áreas, por exemplo aprendizado profundo para os modelos a serem usados, controle de concorrência quando esta for implementada e o uso de índices. Também é um sistema que herda as dificuldades e desafios dessas diferentes áreas, seja com buscar algoritmos melhores para os modelos de aprendizado profundo ou conseguir uma forma mais eficiente de lidar com dados em grande escala.