Progressive Indexes

Timbó Holanda, P.T.

CHAPTER 1

Introduction

1 Data Analysis

Data Analysis is the process where a scientist extracts valuable knowledge (e.g., data correlation, useful patterns, market trends) and uses this information to make decisions.

Using the car industry as an example, car engineers generate random deformations on car shapes to better understand how design changes affect aerodynamics. The actual process of analyzing the generated simulation data is done by data scientists [33]. For example, the data scientists might learn that a slightly different position of the car's rear-view mirror might significantly impact its aerodynamics. This process consists of generating random deformations to a car shape (i.e., slightly changing x-y-z positions of the car model) and running it through computational fluid dynamic simulations. Each simulation generates raw files consisting of many gigabytes of data. Due to the data size, the data scientist must apply interactive data analysis techniques to learn which modifications in the car's shape improve its aerodynamics.

One major problem with this approach is that each simulation takes many hours to be executed. One promising solution is to train a machine learning model to generate the simulated data when receiving a car shape as input. Such a model is trained over many gigabytes of already executed simulations and eliminates the necessity of running simulations for new deformations [20, 19]. This type of data analysis pipeline is relatively common in the industry and can be summarized into three main steps:

1. **Pre-Processing:** In this step, data is generated, cleaned, and loaded in a database management system (DBMS).
2. **Interactive Data Analysis:** In this step, the data scientist explores the data sets to gain insights about the data.
3. **Machine Learning Driven Analysis:** A machine learning model is created to accelerate data generation, classification, and prediction.

In this thesis, we focus on optimizations for Interactive Data Analysis. In the following sections of this chapter, we will introduce this topic and present the research questions explored in this thesis.

2 Interactive Data Analysis

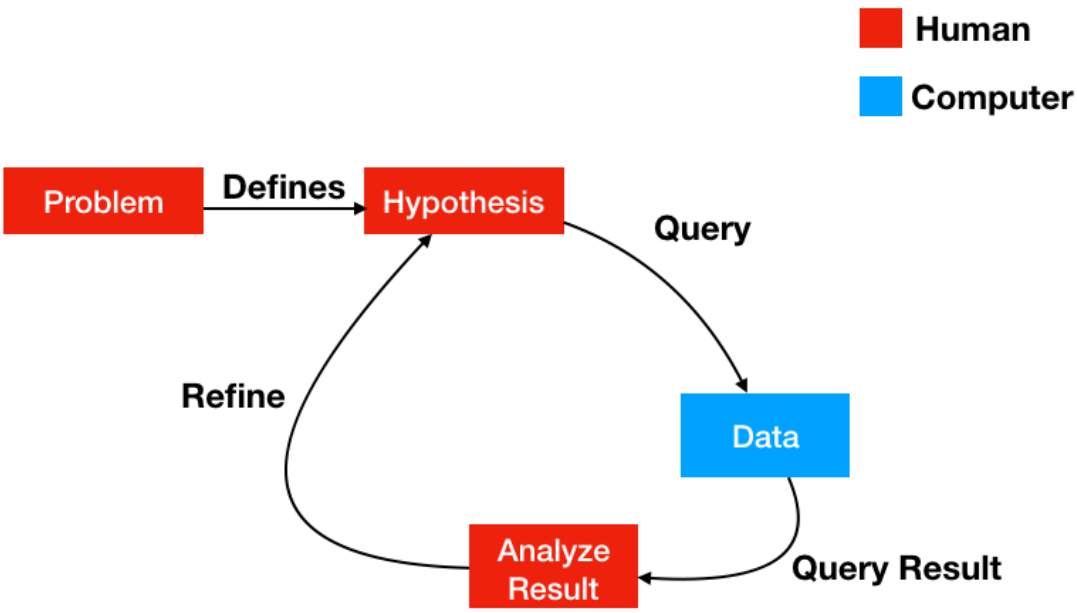

Figure 1-1: Interactive Data Analysis Workflow

Data scientists perform exploratory data analysis to discover unexpected patterns in large collections of data. This process is done with a hypothesis-driven trial-and-error approach [52]. Figure 1-1 depicts the classical interactive data analysis problem. The data scientists derive hypotheses and test them by querying segments that could potentially provide insights. With this result, they refine their original hypotheses and either zoom in on the same segment or move to a different one depending on the insights gained.

In this workflow, the data scientist is always in the loop and depends on fast query responses to perform interactive data analysis. The study by Liu et al. [41] shows that any delay larger than 500ms (the "interactivity threshold") significantly reduces the rate at which users make observations and generate hypotheses.

When dealing with small data sets, providing answers within this interactivity threshold is possible even when only performing full scans on the data. However, exploratory data analysis is often performed on larger data sets as well. For example, the SkyServer project [56] which maps the universe, consists of many terabytes of data. This project has many interactive queries, with data scientists checking different hypotheses on small segments of the sky. Due to the massive amount of data, answering these queries under the interactive threshold by performing only full scans is unfeasible.

An essential optimization that these highly selective queries require is the exploitation of secondary index structures. Depending on the query's selectivity, an index structure can diminish the execution time in orders of magnitude, allowing for responses in interactive times.

2.1 Index Creation Problem

Index creation is one of the major difficult decisions in database schema design [15]. Based on the expected workload, the database administrator (DBA) needs to decide whether creating a specific index is worth the overhead in creating and maintaining it. Creating indexes up-front is especially challenging in exploratory and interactive data analysis, where queries are not known in advance, workload patterns change frequently, and interactive responses are required. In these scenarios, data scientists load their data and immediately want to start querying it without waiting for index construction. Also, it is not certain whether or not creating an index is worth the investment at all. We cannot be sure that the column will be queried frequently enough for the large initial investment of creating a full index to pay off.

Despite these challenges, indexing remains crucial for improving database performance. When no indexes are present, even simple point and range selections require expensive full table scans. When these operations are performed on large data sets, indexes are essential to ensure interactive query response times. Two main strategies aim to release the DBA of having to choose which indexes to create manually. The first step at automatizing the index creation problem was self-tuning tools. These tools perform offline (i.e., indexes are created when the database is not being used) and online (i.e., indexes are created while queries are executed) full index creation.

The second step was adaptive indexing techniques which perform incremental index creation (i.e., indexes are created partially during query execution).

Self-Tuning Tools

Self-tuning tools [1, 14, 58, 23, 13, 11, 44, 53] are pieces of software that gives hints to the Database Administrator (DBA) on which indexes to create for a database. Those hints are an attempt to find the optimal set of indexes given a query workload. These tools consider the benefits of having an index versus the added costs of creating the entire index and maintaining it during modifications to the database.

Self-tuning tools are very successful in traditional OLAP/data warehouse scenarios (e.g., Producing reports). In these scenarios, there is a lot of workload knowledge, the workloads do not change regularly, and the systems are idle off-working times to perform full index creation.

However, these tools are not suitable for index creation in exploratory data analysis due to four main reasons. (1) They require a priori knowledge of the expected workloads. (2) They do not quickly adapt to frequently changing workloads. (3) They require idle time to perform full index creation. (4) The data scientist must take the database administrator's role in analyzing the hints produced by the tools to decide which indexes should be created ultimately.

Adaptive Indexing Techniques.

Adaptive indexing techniques such as database cracking [36, 21, 50, 49, 26, 35, 37, 47, 46, 25, 34, 29] are a more promising solution for the index creation problem in interactive data analysis. They focus on automatically and incrementally building an index as a side effect of querying the data. An index for a column is only initiated when it is first queried. As the column is queried more, the index is refined until it eventually approaches a full index's performance. In this way, the cost of creating an index is smeared out over the cost of querying the data many times, though not necessarily equally, and there is a smaller initial overhead for starting the index creation.

Although adaptive indexing techniques can alleviate the shortcomings of self-tuning tools, they introduce new issues that have not been completely tackled yet. (1) The first query cost can be much higher than a simple full scan cost. (2) There is no guarantee of robustness, and (3) There is no guarantee that the index will eventually converge to a full index (i.e., index all points in the dataset).

First Query Cost. When executing a query on a column for which no index

has been created yet, a full copy of the data is performed to a secondary index structure. After completing the copy, the data is then partitioned into one or more pieces depending on the used adaptive technique. This process incurs a much higher cost than simply scanning the data to answer the query.

Robustness. In general, adaptive indexing only refines pieces that are accessed. When the same piece is constantly requested, its access time becomes similar to a full index. However, as soon as the data scientist decides to query a less refined piece, the query performance degrades, causing performance spikes.

This scenario is highly undesirable for the user since it brings unpredictability to the query response time. Similar queries (i.e., queries that inspect the same amount of data in a column) can have widely different response times.

Convergence. In general, only accessed data points are added to the index structure. Although adaptive indexing can achieve near full index response time when pieces are sufficiently refined, it will not guarantee a full index response time to any filter predicates unless all data points were used as filter predicates.

2.2 Research Questions

Our research aims to investigate how indexes can be created and refined in a similar process as adaptive indexing while inflicting a low indexing penalty on the initial queries, enforcing a predictable query response time and with guaranteed full index convergence. Ultimately, we envision an indexing technique, called *Progressive Indexing* that mitigates these drawbacks from adaptive indexing by performing a more fine-grained refinement and progressively converging to a full index.

We define our main research question as follows:

Research Problem 1 How can we create/refine indexes during query execution with a low impact over initial queries, predictable query response time, and guaranteed full index convergence?

Research Problem 2 How can we create progressive indexes for queries with filters on multiple attributes?

Research Problem 3 How can we update progressive indexes while keeping predictable query performance and guaranteed full index convergence?

3 Our Contributions

This thesis describes how to create progressive indexing techniques for both unidimensional and multidimensional index structures. For each, we explore the current state-of-the-art on adaptive indexing and attempt to improve the characteristics defined in our main research question.

Its main contributions are as follows:

- **Progressive Indexing (Chapter 3).** We alter various sorting algorithms (i.e., Quicksort, Radixsort - Most Significant Digit, Radixsort - Least Significant Digit, and Bucketsort Equiheight) to work progressively following a pre-defined indexing budget.
- **Greedy Progressive Indexing (Chapter 3).** We define a cost model for each Progressive Indexing algorithm that allows for automatic selection of the indexing budget.
- **Progressive KD-Tree (Chapter 4).** We propose a multidimensional progressive indexing, the Progressive KD-Tree, that progressively builds KD-Trees for queries with filters in multiple dimensions.
- **Greedy Progressive KD-Tree (Chapter 4).** We define a cost model for our Progressive KD-Tree algorithm, allowing for an automatic selection of the indexing budget.
- **Progressive Merges (Chapter 5).** We define a new progressive indexing technique used to merge appends into progressive indexes.

4 Structure and Covered Publications

Chapter 3 describes the progressive indexing techniques for Quicksort, Radixsort - Most Significant Digit, Radixsort - Least Significant Digit, and Bucketsort Equiheight in their traditional and greedy formats. This chapter is based on the following papers:

- **Progressive Indexes: Indexing for Interactive Data Analysis [32].**
  Pedro Holanda, Mark Raasveldt, Stefan Manegold and Hannes Mühleisen
  46th International Conference on Very Large Databases (VLDB 2020)

- **Progressive Indices – Indexing Without Prejudice [28].** Pedro Holanda, 44th International Conference on Very Large Data Bases (VLDB 2018, PhD Workshop)

Chapter 4 presents both traditional and greedy versions of multidimensional progressive indexing, a technique based on quicksort and KD-Trees. This chapter is based on the following papers:

- **Multidimensional Adaptive & Progressive Indexes [43].**
  Matheus Nerone, Pedro Holanda, Eduardo Almeida and Stefan Manegold
  37th IEEE International Conference on Data Engineering (ICDE 2021)

- **Cracking KD-Tree: The First Multidimensional Adaptive Indexing [31].**
  Pedro Holanda, Matheus Nerone, Eduardo Almeida, and Stefan Manegold, 7th International Conference on Data Science, Technology and Applications (DATA 2018, EDDY)

Chapter 5 describes Progressive Merges, a technique developed to progressively merge batches of appends into progressive indexes without impacting the predictability of the queries. This chapter is based on the following paper:

- **Progressive Mergesort: Merging Batches of Appends into Progressive Indexes [30].** Pedro Holanda and Stefan Manegold, 24th International Conference on Extending Database Technology (EDBT 2021)

Finally, Chapter 6 summarizes the work done in progressive indexing, discusses its challenges, and provides future work.