Universidade do Minho
Escola de Engenharia
Departamento de Informática

Rogério António da Costa Pontes

Benchmarking a Linear Algebra Approach to OLAP
Master Thesis

November 2015

Universidade do Minho
Escola de Engenharia
Departamento de Informática

Rogério António da Costa Pontes

Benchmarking a Linear Algebra Approach to OLAP Master Thesis

Master dissertation
Master Degree in Computing Engineering

Dissertation supervised by
José N. Oliveira
José O. Pereira

November 2015

ACKNOWLEDGEMENTS

This work would not have been possible without professor José Nuno Oliveira trusting me to bring to reality his theoretical foundations for OLAP, guiding me through the development of the dissertation and actively dedicating himself to improving my work.

Professor José Orlando Pereira and professor Rui Oliveira have been fundamental for integrating the dissertation in the project LeanBigData, giving me the opportunity to work in their HASLab group and aiding in several ways the development of the project. I'm also thankful to the elements of HASLab for their constant assistance, specially to Miguel Matos for his insights in several topics and aiding me on the experiments evaluation.

I'm ever grateful for the hard work that everyone in my family endured through their lives to support me and showing me the importance of perseverance. A journey with company is always more pleasant with someone special, Vera Matos thank your for walking alongside me.

This research has been partially funded by the European Commission under project LeanBigData (FP7-619606).

ABSTRACT

Data warehouses and Online Analytical Processing (OLAP) systems are a critical part of any successful enterprise, they provide essential insight to several aspects of a business, such as the customers preferences or the revenue being generated. This reliance keeps getting stronger as data grows and the tools that analyses it become more accessible and efficient Chen et al. (2014); Thusoo et al. (2010). SQL fails on providing a formal semantics for the most common operations used on OLAP systems that deal with the quantitative side of data (aggregating values with functions), unlike Online Transaction Processing (OLTP) databases, that deal mostly with the qualitative side. While a formal framework was defined in Lenz and Thalheim (2009), a novel approach based on linear algebra (LA) Macedo and Oliveira (2014), deals with the qualitative side as well the quantitative side. Additional this algebra is capable of expressing quantitative analysis solely in term of matrix operation such as multiplication, transposition etc. As the parallelization theory of such operations is well acknowledged, the purpose of this work is to benchmark the approach using realistic data on a distributed environment. The main idea is to test whether the parallelism inherent in the LA scripts presented in the paper materializes in real-life big-data analysis. From the developed work it seems there is an increase of efficiency when the matrix operations can be computed lazily, however several topics both in terms of algebra transformation and implementation are still open to be explored.

RESUMO

Data warehouses e Sistemas de processamento analítico Online (OLAP) fazem parte de todas as empresas com sucesso. Estas tecnologias fornecem informação essencial sobre vários aspetos do negócio, tal como a preferência dos consumidores e o lucro que é gerado. A dependência das empresas nestas tecnologias continua a crescer á medida que a quantidade de dados existente aumenta e as mesmas ferramentas ficam mais acessíveis e eficientes Chen et al. (2014); Thusoo et al. (2010). SQL não possui uma semântica formal para a maior parte das operações utilizadas em sistemas OLAP. Estes sistemas são especializados para tratar com os dados de forma quantitativa, agregando valores com funções, ao contrário de systemas de processamento de transações online (OLTP), que lidam principalmente co o lado qualitativo dos dados. Existe uma framework formal que define o lado quantitativo em Lenz and Thalheim (2009), mas no entanto uma ma nova abordagem a sistemas OLAP baseada em álgebra linear foi recentemente publicada, demonstrando como a análise quantitativa de dados pode ser expressa unicamente com operações matriciais tais como multiplicação, transposição etc. Sendo a teoria da paralelização de tais operações bem conhecida, este estudo tem como objetivo avaliar o desempenho de uma solução baseada neste teoria utilizando dados realísticos num sistema distribuído. O objetivo principal é verificar se o paralelismo inerente nos "scripts" de álgebra linear apresentados no referido artigo se materializa de facto no terreno, num contexto de "big-data analysis". Dos resultados obtidos existe um ganho de performance quando as matrizes são computadas de uma maneira ´"Lazy", no entanto existe um vasto leque de questões que devem ser abordados quer em termos de tranformação entre algebra quer em termos de computação de matrizes.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

List of Tables

List of Tables

LIST OF ACRONYMS

LA	Linear Algebra
TLA	Typed Linear Algebra
DBMS	Database Management System
RDBMS	Relational Database Management System
OLAP	Online Analitical Processing
OLTP	Online Transactio Processing
ROLAP	Relational Online Analitical Processing
MOLAP	Multidimensional Online Analitical Processing
HOLAP	Hybrid Online Analitical Processing
LAOLAP	Linear Algebra Online Analitical Processing
ORC	Optimized Row Column
CSV	Comma Separated File
HDFS	Hadoop Distributed File System
TPC	Transaction Processing Performance Council
ORC	Optimized Row Column

Part I

INTRODUCTORY MATERIAL

1 INTRODUCTION

1.1 CONTEXT AND MOTIVATION

Data have grown tremendously in size over the last few years along with the importance of analyzing them to obtain timely and relevant knowledge. If companies want to be competitive they have to be able to harness all the information they can gather from their clients, competitors or any other relevant sources, so as to decide what their best course of action is for the future. An example of success is Harrah's Entertainment, a Las Vegas gaming corporation which has increased revenue by changing their customer practices based on insights obtained from their customer-centric data warehouse (Watson and Wixom, 2007).

The relevance of data analysis spreads to many aspects of our lives back in 2009 an influenza outbreak started in Mexico. It took several months until it was considered epidemic, in spite of the government's best efforts on controlling the affected areas the virus spread globally. Google was able to gather timely information about the outbreak due to users entering different search terms during the outbreak. From this experience they developed an important line of defense against future influenza outbreaks, a monitoring system capable of detecting an outbreak with one day lag (Ginsberg et al., 2009).

Data analyses centers around *aggregating data*, which is by definition any process that gathers information and expresses it in a summary form. The first database systems were capable of handling small amounts of data and calculating simple aggregations such as minima, maxima or sums. As enterprises started to use databases to store their data, the existing quantity of data grew significantly, typically from some terabytes to petabytes. Accompanying the growth of data, queries submitted to operational databases that would take a few minutes to analyze would then take hours or even days to complete (Codd et al., 1993). Since the databases were no longer capable of handling this increase in volume of data and by noticing that there were two types of operations commonly used in databases two distinct kinds of application were defined:

- on-line transaction processing (*OLTP*) systems focusing on fast transaction processing that addresses day-to-day important information;
- *Date Warehouses* that store data from several different sources, that might be an OLTP, and provide reports and analysis.

Inside a Warehouse there is usually another system encompassing so-called on-line analytical processing (**OLAP**) applications. These are responsible for managing aggregated data, executing complex long lasting queries and helping the data analyst when taking decision based on the summarized information. OLAP gives this support to the analysts by answering questions about the data such as "what-if" and "why" since they can easily query, skim and visualize a large quantity of information (Chaudhuri and Dayal, 1997).

The amount of data that is being currently produced by companies such as Google or Facebook led to a new concept, that of *Big Data*, a huge amount of data coming not only from direct human activity but also from other autonomous sources of information, for instance sensors in smart grids, that can not be captured, managed and processed by the traditional technology. With the advent of Big Data enterprises are looking for new solutions able to handle and take advantage of the available information to have an edge in business (Chen et al., 2014).

This scenario calls for parallel and distributed solutions. An emerging open-source solution, *Hadoop* (Dean and Ghemawat, 2004), provides a framework capable of managing and interconnecting large quantity of machines who cooperate to address Big Data challenges. Supposedly, in 2011 Facebook had 2000 machines storing 21PB of data[1], by creating a distributed file system (HDFS) on top of such a network of machines. Hadoop also offers a computational abstraction, *MapReduce* (Dean and Ghemawat, 2004) that allows software developers to analyze the data stored in HDFS in a simple and distributed way to improve the overall computation performance.

As the success of companies become more dependent on these technologies it becomes increasingly important to have a fast and correct solution. It becomes difficult to have a solution that can be trusted when no formal semantics are being used to describe the quantitative side of data analytics. One definition that works on top of relational algebra has been given in Lenz and Thalheim (2009) but they still work on top of the relational algebra and do not provide a logical framework to prove the OLAP operations with simple mathematical expressions. Another proposal to formalize this issue is tackled in Macedo and Oliveira (2014) that provides a novel approach based on linear algebra (LA) capable of not only executing the same operation on OLAP systems (data cube, roll up, cross tab) but also provide formal proofs both on the quantitative side and qualitative side. This theory works by typing matrices with the columns attributes and using matrix operations to compute the queries. Additionally it provides a theoretic distributed incremental framework for computing the queries. The approach proposed by Macedo and Oliveira (2014) provides a new niche of solutions worth exploring in this field concerning the practical viability of such new ideas. Thus the idea, central to the work reported in this document, explore this new niche and benchmarking the findings on a cluster of distributed machines.

1 Source: http://hadoopblog.blogspot.pt/2010/05/facebook-has-worlds-largest-hadoop.html consulted November 3, 2015.

1.2 MAIN AIMS

This dissertation focus on applying the LA theory in a cluster of machines using the Hadoop framework. For that several steps have to be accomplished: first find efficient representation solutions for the matrices based on the current state of the art on matrix computation; secondly apply the theory defined in Macedo and Oliveira (2014) to more world scenarios based on the TPC-H quereis, thirdly use the Hadoop framework to create the benchmark experiments and finally assess the results. There is an extra step, made in this work, which is just a small part of future work that is the conversion between SQL and typed Linear algebra. This step allow us to have a formal definition for implementing the queries.

1.3 STRUCTURE OF THE DISSERTATION

Chapter 2, reviews state-of-the-art research in traditional database systems, warehouses, OLAP engines and current developments on the Big data trend. Chapter 3 gives a quick introduction and summary on how linear algebra can be used to solve some simple SQL queries. It also addresses the important topic of how the matrices used to calculate aggregates can be effectively stored and what algorithms can be used to calculate aggregates from these storage formats directly. Following this introductory parts, we start tackling in Chapter 4 some existing representation for matrices and work to improve them to our needs. Afterwards, in Chapter 5, we give a deeper understanding how the Hadoop framework works, how it was used and the matrices are divided in the cluster. Before the last chapter we provide the initial work to translate SQL to LA in Chapter 6 so that we can correctly translate the queries used to benchmark. In the last Chapter 7 we present the results of two benchmarks on the work developed.

2

STATE OF THE ART

2.1 INTRODUCTION

This sections tackles several topics: first it starts by introducing the relational model with some of the most elementary operations; afterwards it presents multiple solutions for a OLAP systems; next it follows an overview of the emerging topic of Big Data and finally presents several industry standard benchmarks that are used to evaluate the many broad systems.

In the early years, most database systems were applications made to solve a specific problem (Codd et al., 1993), users had to know the internal structure of the application (Codd, 1970) and a large data set was considered to go from several megabytes to many terabytes (Graefe, 1993). In 1970 E. F. Codd set the standard by introducing relational algebra, (Codd, 1970) thus facilitating end users with a language they could use without understanding the underlying technology. According to relational algebra, data is stored in a table that represents a subset of the relation, each column designates a domain and each line is a record containing a tuple of the relation. For instance the next table represents the quaternary relation, Users, where the first value is the first name, the second the city he lives in, the thirds contains the gender and the forth his age.

Name	City	Gender	Age
Peter	Dublin	M	30
Kyle	Manhatan	F	32
John	New York	M	87
Kyle	Manhatan	F	24
John	Perth	M	50
John	Lisbon	M	28

Table 1.: Relation Users

Most Relational DBMS store the table in a row oriented format while others store column by column, usually known as Column-oriented database system. Row store architecture provide a better performance for writing since with a single write it is capable of adding new records. Its opposite, column oriented, are based on the premise that most queries require only certain columns and by laying out columns sequentially in memory (Stonebraker et al., 2005) they are capable of reading data more efficiently, thus they are read optimized. Column oriented databases always end up stitching together the columns of a table to recreate the result row and return it to the user and even though some are capable of joining tables without materializing the rows until the final result is obtained (late materialization) only some traditional algorithms used by RDBMS are explained below. The algorithms explained focus on joining tables and calculating aggregations since this are two of the most common operations for summarizing information.

2.2 RDBMS ALGORITHMS

RDBMS rely on two main operations sorting and hashing since most queries need to bring items that are alike together. For instance joining two table is bringing rows that have a common value together and can be done with merge-join or hash-join. The algorithms explained in this section take a naive approach by discarding important implementation details such as main memory size, the time spent in IO, page size, etc. For a more detailed explanation the literature review (Graefe, 1993) provides a deeper explanation, and the third book of “The Art of computer programming” (Knuth, 1997/98) analyzes some algorithms even more accurately. The Youtube channel of professor Jens Dittrich[1] contains a visual explanation of most algorithms and the inner workings of a database. Much of this section in respect to RDBMS is based on Graefe literature review. Three core algorithms used in RDBMS will be presented and small variations depending on if it is a join query or a aggregation query.

The first algorithm uses two **Nested loop** to perform a join of two tables with a common column. This algorithm is known as **Nested loop join** When joining two tables that share a common domain, for instance table A as domain D as a foreign key to table B the first loop, the outer loop iterate through the elements from table A while the second loop, inner loop, goes through table B. If an element from the outer loop matches any of the inner loop then join both tuples in the resulting table. This simple algorithms has very few improvements, in one-to-one match operation the inner loop can stop if a match is found, but aside from this specific improvement all the other optimizations are made in terms of memory access. One possible optimization might be in the inner loop if the column of B is indexed, instead of searching the table sequentially, one can search in the index, thus reducing the inner loop time. In the worst case, where no optimization is possible, if table A as N elements and table B as M then this algorithm performance is $\mathcal{O}(N * M)$, if both tables had the same size K this would be $\mathcal{O}(K^2)$.

When aggregating values of different tables first a **Nested loop join** is performed. On the resulting table it is performed another nested loop that iterates through the table and whenever there is a match between the item in the inner loop and outer loop then calculate the aggregate function and store it. The algorithm must somehow keep track of the aggregation already calculated so that no aggregation in computed multiple times.

1 https://www.youtube.com/user/jensdit

The next algorithm is highly dependent on sorting and when joining tables is commonly known as **Merge-Join**. Since sorting as a deep impact on the performance, a simple understanding of how it is done is required. Most table sizes have always been and will always be bigger than the available space in main memory, thus the actual algorithm used for sorting is **external merge sort**. External merge sort starts by dividing the initial file in runs and afterwards merging this runs until the final sorted file is obtained. The creations of runs can be done in two different ways the first is loading part of the file that fits in memory, sorting it, usually with quick-sort and then write to disk again. Repeating this process until all the file is read. Another method of generating runs is with replacement selection, with this method items are organized in a priority heap with size N. Initially N items are read from the file to the priority heap, afterwards the smallest item from the heap is written to a run. Then read a new item from the original file to the heap. Again take take the smallest value from the heap, if this item is bigger than the last item written to the run, write it in the run file too, otherwise take it of the heap and put in a list for another run. Repeat this processes until the heap is empty, when this happen take the elements from the list to the heap and start another run file. Continue with this algorithm until all of the file is divided in runs. Afterwards merge the runs as usually. On the off chance the input size is small enough to fit in memory an in memory algorithm can be used to sort the data, such as quick-sort.

Merge-Join sorts both tables by the domain they have in common, afterwards, take the first values of each table, a from table A and b from table B. If a is the same as b then there is a join, while a is less or equal then b move through the elements in A, if the element a is bigger than the current element in b then move to the next element in B and start comparing with the next elements in A. The following pseudo code illustrate this algorithm.

Algorithm 1 Merge-Sort

```
Require: Table $A$, Table $B$, Merging Dimension $D$
Ensure: Merged tables
  $SA \leftarrow sort(A, D)$
  $SB \leftarrow sort(B, D)$
  $Q \leftarrow Empty$
  $i \leftarrow 10$
  for $a$ in $SA$ do
    $b \leftarrow head(SB)$
    while $a \leqslant b$ do
      if $a = b$ then
        $Q.add(join(a, b))$
      end if
      $a \leftarrow head(SA)$
    end while
  end for
  return $Q$
```

The presented algorithm must be changed if multiple attribute values exist, requiring several passes in the inner loop. Even though there are two loops as in the Nested-loops join, since the tables are sorted. each one is scanned only once. If the tables are pre-sorted maybe due to a previous operation or because an index is used then the cost of the operation is just the cost of merging both tables. Nonetheless, in the most general case the tables must be sorted and that makes the execution time being most used in sorting which usually takes $\mathcal{O}(n * \log n)$ for each table (Mishra and Eich, 1992).

When aggregating using a sort based algorithm, first the table is sorted by the aggregation attributes and then a simple sequential scan can easily calculate the aggregation result.

As an example of this algorithm take the following tables, the first is a projection of the original table sorted by Name and City the second is the maximum aggregation grouped by name and city:

Name	City	Age
John	New York	87
John	Lisbon	28
John	Perth	50
Peter	Dublin	30
Kyle	Manhattan	24
Kyle	Manhattan	32

Table 2.: Table sorted for aggregation

Name	City	Age
John	New York	87
John	Lisbon	28
John	Perth	50
Peter	Dublin	30
Kyle	Manhattan	32

Table 3.: Aggregation result

As can easily be seen by this small example this aggregation has the same result as of running a unique query on the same attributes. In traditional databases, aggregation and duplicate removal have been implemented in the same module due to their close similarity.

Lastly, the final algorithm is based on Hashing and even though its implementation is conceptually easy, there are some details that one has to be careful when the hash table created does not fit in memory and an **Hash table overflow** occurs. This problem is usually solved with **Hybrid hashing**. In relational joins, **Simple hash join** chooses one table from the join, hashes the attributes of the domain used to join the tables turning them to the hash table keys and stores either the full record or the record-id as the value of the hash. For the table not hashed, iterate through the elements and probe the hash table when there is an value match then create a join tuple.

Algorithm 2 Simple hash join

```
Require: Table $A$, Table $B$, Merging Dimension $D$
Ensure: Merged tables
  $Q \leftarrow Empty$
  $hashTable \leftarrow empty$
  for $a$ in $A$ do
    $hashTable.put(hash(a, D)$,a)
  end for
  for $b$ in $B$ do
    $keygetshash(b, D)$
    if $hashTable.get(key) = b$ then
      $Q.add(join(hashTable.get(key, D), b))$
    end if
  end for
  return $Q$
```

For selecting the table to create the hash table, the algorithm could use the table with the least number of distinct values, but since such information is not always available algorithms usually choose the smallest table. This is especially efficient if the hash table fits in main memory. Hash tables performance depends mostly on how evenly the hash function distributes the keys so if table A as n size and table B as m size the join operation has a complexity of $\mathcal{O}(n * m)$. **Simple hash join** is not the only hash based algorithm, other possible variations are Hash-Partitioned Joins, Grace Hash Join method, etc (Mishra and Eich, 1992). An algorithm based on Hashing for computing an aggregation iterates through the input, hashing the values and inserting them in the hash table. If an element is already present in the hash table when inserting them it is possible to perform the aggregation function.

2.3 OLAP

As databases became widely used in enterprises, their data sets in 1990 went from terabytes to petabytes. Not only the size of information grew but business become more dependent on the analysis provided by them to the point of making or breaking enterprises (Watson et al., 2001).

The existing technology could not cope with the increasing complexity, so a division started to appear, a database with the most up to date information where several transactions were executed per second and another database with the historical information of the Enterprise where the only operations made were loading of data and reading it. The warehouses contained the data but were still based on the traditional RDBMS that weren't capable of handle the complex multidimensional queries being issued.

In 1993 E.F.Codd published a new paper that defined an emerging tool, OLAP, that did not replace the existing databases but complemented them by working with the historical data and respond to the questions "what-if" and/or "why" (Codd et al., 1993). Not only did he defined what a OLAP tool is but set twelve rules to evaluate an OLAP solution.

Grafae (Gray et al., 1997) defines a new format to view aggregates and helped shape data analyses by introducing the concept of **Data Cube** and the operations possible to do in it. In addition it categorized three types of aggregation functions and provided the first yet naive algorithm to compute the **Data cube**. The cube is the power set of a N-dimensional aggregation. For instance the Cube of a sum aggregation of table 1 grouping on the three columns Name, City, Gender generalized to the domains A,B,C produces the following aggregations **ABC**, **AB**, **AC**, **BC**, **A**, **B**, **C** and *all*, where *all* is the empty group-by. This cube is represented in table 4 where the group-bys follow the enumerated order.

Name	City	Gender	Age
Peter	Dublin	M	30
Kyle	Manhattan	F	56
John	New York	M	87
John	Perth	M	50
John	Lisbon	M	28
Peter	Dublin	all	30
Kyle	Manhattan	all	56
John	New York	all	87
John	Perth	all	50
John	Lisbon	all	28
Peter	all	M	30
Kyle	all	F	56
John	all	F	24
John	all	M	78
all	Dublin	M	30
all	Manhattan	F	56
all	New York	M	87
all	Perth	M	50
all	Lisbon	M	28
Peter	all	all	30
Kyle	all	all	56
John	all	all	165
all	Dublin	all	30
all	Manhattan	all	56
all	New York	all	87
all	Perth	all	50
all	Lisbon	all	28
all	all	M	195
all	all	F	56
all	all	all	251

Table 4.: Data cube sum of 1

Two distinct solutions came to calculate and implement the data cube (Chaudhuri and Dayal, 1997), one based on DBMS and is known as ROLAP, other took a different approach and store the a N-Cube as a N-dimensional array and are known as MOLAP (Multidimensional OLAP). ROLAP servers were placed between the warehouse end server that contained all the data usually in a Star-schema or snowflake and the Client. ROLAP takes advantage of the mature and efficient RDBMS but since OLAP queries do not always translate nicely to SQL queries these are not the most efficient engines. On the other hand MOLAP servers provide a multidimensional view of the data by mapping multidimensional queries to a multidimensional storage engine. These are usually seen as having a better performance than ROLAP but have a more complex process of extracting, transforming and loading (**ETL**) data. Additionally MOLAP storage is naturally sparse by growing exponentially with the number of dimensions and the situation becomes worse when the original data is itself sparse thus leading to a significant waste of memory (Hasan et al., 2007). Another OLAP strand exists called Hybrid OLAP that aims to use the best of both solutions.

Regardless of the underlying storage there is one concept similar to both, the cube lattice. The group-bys that make the cube can be partially ordered in a directed acyclic graph (DAG). Every node in the graph represents a group-by queries and each node is linked by a direct edge with every other node that contains a similar group-by but without one grouping attribute.

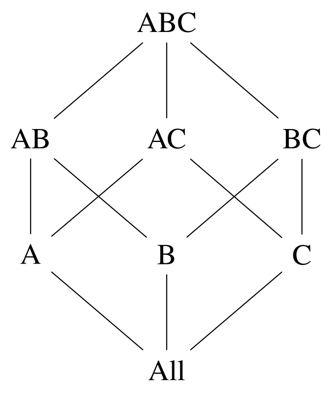


Figure 1.: Cube lattice

The highest node is usually called the root, it has the highest detail and by going down the lattice the queries become more specialized until arriving to t he lowest query ALL. As the number of dimensions increase so does the cost of computing the total cube and independently of the technology this is a costly process, hence it is common to select a proper subset of the data cube, precomputed it and store it for further use. This process is called "data cube implementation" and is divided in two subjects, computing the cube and selection of the cube (Morfonios et al., 2007). Since on this dissertation the focus is on an alternative way of computing the cube, the following subsections give an overview of some algorithms used in ROLAP and only one algorithm for MOLAP.

2.3.1 *ROLAP algorithms*

The first algorithm for computing the data cube 2^D was presented in (Gray et al., 1997) and was based on a RDBMS. The algorithm simply calculated each group-by independently, thus scanning the original fact table each time. After all queries are calculated then unite each result and create the cube. This algorithm is highly inefficient, grows exponentially with regards to the number of grouping dimensions. A cube with N dimensions would take 2^{N-1} scans and aggregations from the original data. Additionally no advantage is taken from results, sorts or hash tables made from other aggregations.

This algorithm was only proposed as a presentation of the data cube. All of the following algorithms create a execution tree T of the cube lattice that is used to compute each node (Morfonios et al., 2007). Different algorithms may generate the execution tree differently by considering different metrics or algorithms, but in all of them, every node of the final execution tree but the root has only one link to another node (the parent), from which it is computed.

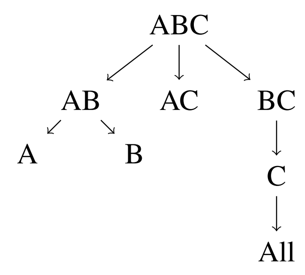
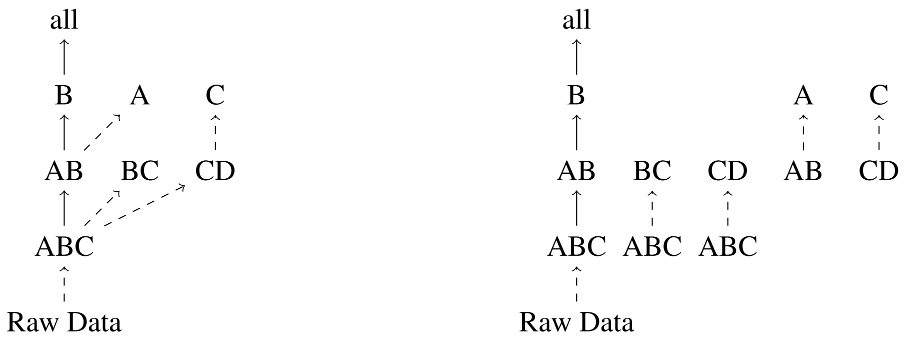


Figure 2.: Example of an execution tree

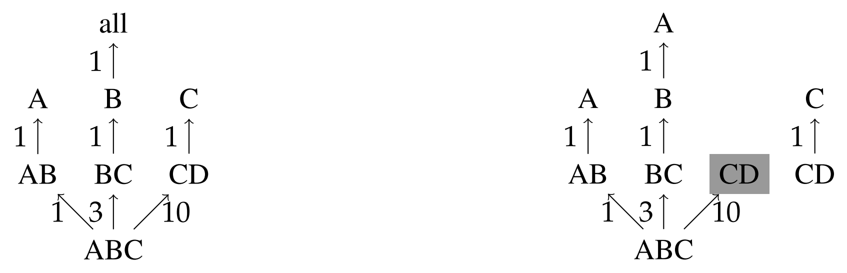
The first algorithm with this approach of creating an execution tree was **PipeSort** (Agarwal et al., 1996). The algorithm starts by creating the cube lattice and in each edge attribute two costs, the cost of computing the edge from another edge in a upper level if it has to sort and if it does not. Afterwards the algorithms transverses the lattice from bottom to the top level by level. The first step replicates each vertex in the $k+1$ level k times and the replicated edges contain the same set of edges as the original and contain the cost if it has to sort. Before preceding to the next level, the altered graph is used to identify the edges that minimizing the sum of edges cost by doing a weight bipartite matching. After going through all levels the original lattice tree is pruned and the execution tree is created. The final step of the algorithm takes the execution tree and transforms it in a set of paths (pipelines), where an edge appears in only on one path and only the first edge of each path has to be sorted. According to the authors this algorithm complexity is $\mathcal{O}(((K+1)M_{k+1})^3)$, M_{k+1} is the number of group-bys in level $k+1$. This algorithm is illustrated in the Figure 3a where it presents the execution tree after pruning the lattice and the parent of a dashed connection must be first sorted to compute the children.



(a) Example of PipeSort execution tree (b) The pipelines that are executed

Figure 3.: Illustration of PipeSort

The next algorithm **Overlap** takes advantage of a property that PipeSort does not. Overlap is based on the idea of partially-matching sort orders which decreases the number of necessary sorts. Given a aggregation sorted by ABC, then the aggregations of AB, AC, BC, A, B and C can be computed from the first aggregation without having to re-sort since they are a subset of the original. The first step of the algorithm chooses a sort order that defines the root of the lattice and this selection is based on heuristics. With the initial sort order defined, the algorithm transverse the lattice created from that order and in each node chooses just one parent, the one which shares the longest prefix. With the execution tree created, each node is attributed a cost, the memory size required to compute it from its parent. This attribution of the cost constitutes the final execution tree. Afterwards the algorithm loops until all nodes are calculated, in each step of the loop, a set of aggregation that have not been processed are selected in a top-down/breadth-first traversal where aggregations with a greater number of attributes and small estimated memory have priority. This selection ends according to memory constraints. The rest of the nodes are put in a "Sort Run" class that are sorted while the selected nodes are computed.



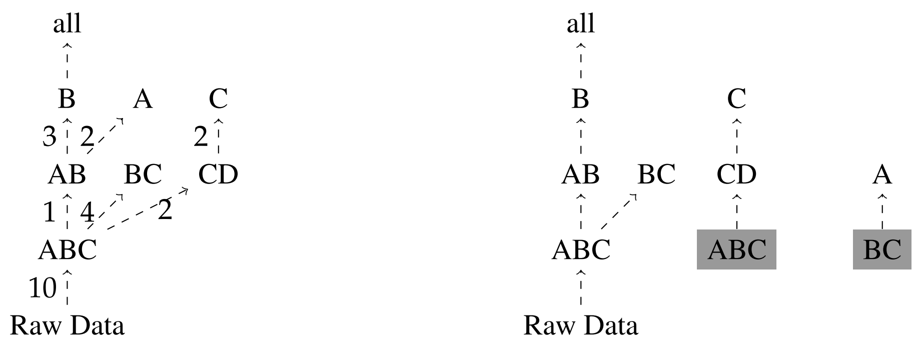
(a) Example of Overlap execution tree

(b) Example of subtrees selection

Figure 4.: Illustration of Overlap

Figure 4a represents the pruned lattice into a execution tree with the costs of computing a node from its parent. It can be seen in the children of ABC that the costs increase from left to right due to not being able share a prefix. The other figure 4b presents the two subtrees selections that fit in memory to be computed. The node with a gray background represents the aggregation that has to be first sorted and does not fit in memory.

Another algorithm presented at (Agarwal et al., 1996) is based on hashing to place tuples that aggregate together on continuous memory positions. **PipeHash** firsts calculates a minimum spanning tree (MST) from the cube lattice, where a node is connected to its parent that has the smallest estimated total size. Since in general all the aggregations in MST cannot be computed in memory the tree must be partitioned. PipeHash started by putting the MST in a work list, then if the entire MST is too big to be able to compute it in memory it divides the original MST in to smaller subtrees. The division into smaller subtrees is made by choosing a subset of the attributes from the root group-by that create the largest partitions that still fit in memory. The partitions are then inserted at the work list and computed using an Hash based aggregation algorithm.



(a) Example of Minimum Spanning tree created (b) Partitions created by dividing on attribute B

Figure 5.: Illustration of PipeHash

Figure 5a contains a MST calculated by PipeHash that can not fit in memory, then the algorithm divides the tree into three subtrees 5b that do fit by partitioning by the attribute B of the root query. From the three subtrees only one contains all the aggregation that contain the attribute B while the others have the rest of the aggregations. The nodes with a gray background represent aggregations that have already been computed when the first subtree that contains B was computed.

There is a modification to the traditional data cube commonly known as the **Iceberg-Cube**, containing group-by tuples with an aggregate value greater than a predefined minimum (*minsup*).

BottomUpCube (BUC) computes the cube by transversing the lattice in a bottom up fashion as the name suggests. **ALL** is the first node computed containing just one tuple and if the value is smaller than the *minsupp* the no other node needs to be computed since they would all be smaller. If the values is greater than *minsupp*, it partition the original table on each dimension of the aggregation. Any aggregation that passes the minimum support condition is passed as input to a recursive call to BUC where it is further partitioned on the remaining dimensions. This type of lattice pruning is possible since if a less detailed aggregation thus not pass the predefined minimum then a more detailed one can not either.

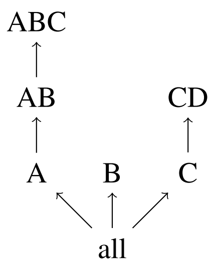


Figure 6.: Example of a possible BUC execution

Figure 6 illustrates a possible execution of a BUC algorithm where almost all nodes are possible to be calculated, the aggregation BC is not present thus the aggregation B thus not pass the minimum support condition.

2.3.2 *MOLAP algorithm*

MOLAP is inherently different from ROLAP due to the data structure used to store the information (Zhao et al., 1997). As mentioned previously, while ROLAP uses the same approach as RDBMS to store information, MOLAP uses a multidimensional array which is conceptually similar to the data-cube. Table 1 contains three dimensions *name*, *city*, *gender* and one measure, *Age*, in a MOLAP engine only the measure of each row would be stored and its position in the multidimensional array would indicate its relation with the dimensions. Due to this type of storage some of the common strategies used on ROLAP algorithms can not be used, such as sorting or hashing to bring tuples alike together. Nonetheless it is still possible to compute an aggregate from another previously computed aggregated instead of the original multidimensional array. The techniques used on the algorithm that will be presented consist on computing as many aggregates as possible while transversing each value of the array just once. In fact due to memory constraints the multidimensional array has to be searched multiples times but the order in which is transversed has a great impact on efficiency.

Common array storage techniques, row major order or column major order, have a different performance when transversing the array, depending on the transversing order. Since browsing the array can be made in many different ways, multidimensional arrays are divided in chunks (Sarawagi and Stonebraker, 1994) thus acquiring uniform treatment in all dimensions. Chunking splits a n-dimensional array into n-dimensional chunks where each chunk has n dimensions. Suppose a table with three dimensions A, B, and C each with 16 elements in each dimensions was imported to a MOLAP engine. This would generate a three dimensional array with the size $16 \times 16 \times 16$. This array can be stored with 64 chunks where each one are of size $4 \times 4 \times 4$ as illustrated in Figure 7.

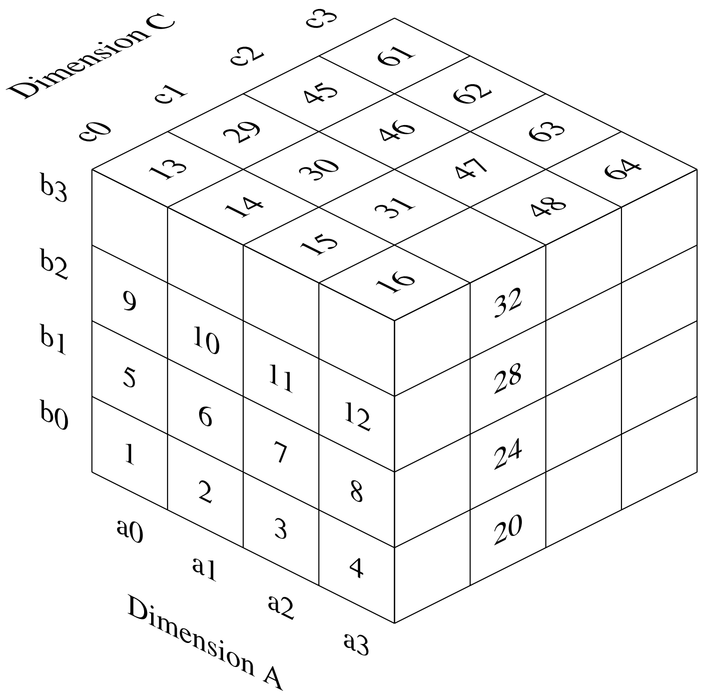


Figure 7.: Three dimensional array extracted from (Zhao et al., 1997)

The numbers in the image represent the chunk number. With the referred image it becomes easy to visualize that to compute the AB group-by, the array is aggregated along the C dimension, for AC group-by it is aggregated on dimension B and for BC it is aggregated along A.

A algorithm to compute the cube of the array, **Singl-Pass Multi-Way Array Algorithm** assumes there is unlimited memory to compute all of the 2^n aggregations. In real systems no such assumption can be made, thus the **Multi-Pass Multi-way Array Algorithm** must be used, but only the first is presented since the second algorithm is very similar to the first but does only compute aggregates that fit in memory in each pass.

Since it is assumed that main memory is no problem a naive algorithm could allocate memory for every chunk and as it reads then it would calculate the group-bys required. Nonetheless a smarter algorithm can be used which allocates only the minimum memory required to compute each group-by. The memory required to compute multiple group-bys in one-pass depends on the order in which the input array is scanned, *dimension order* is a row major logical order that indicates how the array will be transversed independently on how it is stored. Suppose then that the array in figure 7 is going to be read in the dimension order ABC meaning it goes from 1 to 64. By reading the first four chunks from 1 to 4 the aggregation of the chunk b0c0 is completed. After computing the chunk b0c0 this can be write out and reassigned to b1c0. By following this train of thought only one chunk is required to compute the BC group-by. To compute AC group-by since there are four different possible combinations when going by ABC order, only four chunks are required to compute the AC group-by. For instance from 1 to 4 parts of the four chunks can be computed a0c0, a1c0, a2c0, a3c0. When the first 16 chunks those aggregates are done and another four can take their place, a0c1, a1c1,a2c1,a3c1. The AB group-by is the one requiring more memory, a total of 16 chunks and the aggregation of each chunk is completed only when all the 64 chunks are read. An optimization can be made to compute the aggregates of A, B and C by computing each aggregation from the previously computed chunks of AB, BC and CD.

To understand the real importance of the dimension order suppose there is a four dimensional array ABCD, where the size of each dimension is 10, 1000, 1000 and 10000. The original article claims that if the cube was transversed in ABCD order it would require only 4MB to compute the entire cube but if it was computed with the dimension order of DBCA then 4GB would be required.

2.3.3 *A Minimal Cubing Approach*

A completely different approach to computing the cube was taken in (Li et al., 2004) which can be seen as an instance of the theory presented at (Macedo and Oliveira, 2014) as they create the LA projections functions but in a different format and instead of working with matrix multiplications they do set intersections on their storage formats. Thus the computations carried out are very similar but have the same purpose. The authors observed that in the scientific domain such as bioinformatics, most data-sets contain a high number of dimensions but a smaller number of tuples than in enterprises environments. Such data-sets may contain over 1000 dimensions with 10^6 tuples while in traditional databases the norm is around 10 dimensions with a higher number of tuples such as 10^9. As an example consider a data-set with 100 dimensions each with 10 unique attributes, the entire data cube may have up to 11^{100} aggregated cells. This amount of aggregations becomes impracticable to compute and store with the algorithms presented until now mainly due to the storage requirements. Not only it is unreasonable to compute so many aggregations but also most OLAP queries use just a small number of dimensions each time since it becomes hard for analysts to understand a high-dimensional space.

The solution proposed, **shell-fragments** pre computes and stores offline a small disjoint set called *fragments* in a different format from all those previously presented until now and for those queries that don't have its aggregation precomputed it computes them using the precomputed fragments in a timely manner. Thus making it a cost-efficient solution by saving space while still having a good response time.

Given a table with n dimensions these are vertical partitioned into disjoint sets, the fragments. A fragment can have any number of dimensions in any order but no two fragments can have a common domain. For each fragment the entire data cube is precomputed and an inverted index is stored. Take for instance a table with 60 dimensions, $A_1, A_2, ..., A_{60}$, this can be partitioned in groups of three, (A_1, A_2, A_3), $(A_4, A_{,5}, A_6)$, ..., (A_{58}, A_{59}, A_{60}), making a total of 20 fragments. The data cube in each fragment is computed by intersecting the inverted indexes in a bottom-up depths-first order in the cuboid lattice. Using table 1 and using a fragment of size 2, the resulting fragments would be (Name,City) and Gender since Age just contain measures then it would be stored in a *ID measure* array where its positions in the array corresponds to the positions stored in the inverted indexes as seen in the following table:

Attribute	TID list
Peter	1
Kyle	2 4
John	3 5 6

Table 5.: Name Inv. index

Attribute	TID list
Dublin	1
Manhattan	2 4
New York	3
Perth	5
Lisbon	6

Table 6.: City Inv. index

Attribute	Intersection	TID List
(Peter, Dublin)	$1 \cap 1$	1
(Peter, Manhattan)	$1 \cap 2\ 4$	$\varnothing$
...	...	...
(Kyle, Dublin)	$2\ 4 \cap 1$	$\varnothing$
(Kyle, Manhattan)	$2\ 4 \cap 2\ 4$	2 4
...	...	...
(John, Dublin)	$3\ 5\ 6 \cap 1$	$\varnothing$
(John, Manhattan)	$3\ 4\ 5 \cap 2\ 4$	4
...	..	...

Table 7.: Fragment (Name,City)

The previous tables exemplifies a part of the process of creating the fragments by showing the inverted index of the domains Name, City and how these are used to compute the cuboid (Name, City) which itself can be used to intersect with other cuboids such as Gender. The computed shell fragment cubes facilitate online computation and have a nice property of being able to discard tuples that have empty sets. Online computation is in itself a simple algorithm that also intersects the tid list to obtain the tuples that correspond to the aggregation query, once the tuples are recreated these can passed to any cubing algorithm to calculate the local data cube. An important aspect of the fragments is in the decision to group dimensions, since it is possible based on the semantics of the data to group dimensions that are frequently used so that their cubes are already precomputed.

2.4 BIG DATA

Information just keeps on growing (Chen et al., 2014), in 2011 the world created and copied around 1.8 ZB data, mostly unstructured. Once again the existing technology in 2000 when this phenomenon had its beginnings was overwhelmed (Russom and IBM, 2011).

This technology gives business a new opportunity to learn more about themselves, how the business is changing and how to plan for the future by recognizing sales, market opportunities or having a better targeted marketing for instance.

Enterprises are not the only sector taking advantage of Big Data, behind the many people that are using it are governmental organizations, educational institutions and health care (Groves and Knott, 2013). The world urges for a solution capable of managing an enormous amount of raw data and extract valuable information out of it in an acceptable time frame. Hadoop is a possible solution by providing a framework of tools to store and analyze large datasets by using at its core a distributed file system, HDFS and a computation model, MapReduce. The framework was developed with the following ideas (Lin and Dyer, 2010):

HORIZONTAL SCALING The workload generated from the large data sets are too much for just one machine thus it is required to have many machines working in cooperation. Two options are available when considering multiple machines, either have a small number of high end servers, Scaling up, or have large amount of commodity machine, Scaling out. When application are more data-intensive then processing it has been shown (Barroso et al., 2013) that low-end machine have a comparable performance to the high-end ones, thus not justifying the price of better hardware.

HARDWARE FAILURE When there is a high number of machine the probability of one of them to fail increases and in the real world hardware failure should be considered the norm. The detection of failure and recovery of a machine without interrupting the other machines must be achieved in order to provide a seemingly stable system.

DATA LOCALITY Since Hadoop is targeted to data-intensive application it is cheaper to move the computation then to move data between the nodes. By moving computation closer to where the data is instead of moving the data, network congestion is minimized and the system throughput increases.

BATCH PROCESSING One of the problem in computing is the bottleneck caused by the slow hardware storage, while CPU performance and disk size have increased over time and this trend seems to keep up. The the time it takes to read, seek and store information on disk does not. Since random access to data is not efficient it is better for computation to be made in such a way it can read data sequentially.

ABSTRACTION Writing software is complex and a distributed system is even harder due to several intricacies such as starvation and deadlock. If the developer is not concerned with system-level details he can focus on the important aspects of computation, *what* computation needs to be done and *how* these will actually be performed.

SCALABILITY If a jobs takes one hour to be completed in one machine it is to be expected that if the same job is divided by two machines it will take half of its time. Even though this is clearly far from reality due to several constraints, a significant improvement is still expected.

The next subsections give an overview of the inner workings of HDFS and MapReduce. Another subsection explain the usefulness of some tools build on top of Hadoop.

2.4.1 *HDFS*

On of the core of the Hadoop framework is the Hadoop Distributed File System (HDFS), made to be deployed on low-cost machines where faults are considered the norms, as such it was developed to be highly tolerant to them.

In order to keep the system coherent two type of entities are used in HDFS, the **NameNode** and **DataNode** which have a client/master architecture. There is one DataNode for each machine that is in HDFS, this entity is responsible for keeping track of the block stored in the corresponding machine and responding to request given by either the NameNode or a client. The NameNode is the master and contains the meta-data of the files, which is where each block of the file is located, user access permissions and additional options such as replication factor. Now it becomes clear that as block sizes decreases the meta-data in the NameNode increases.

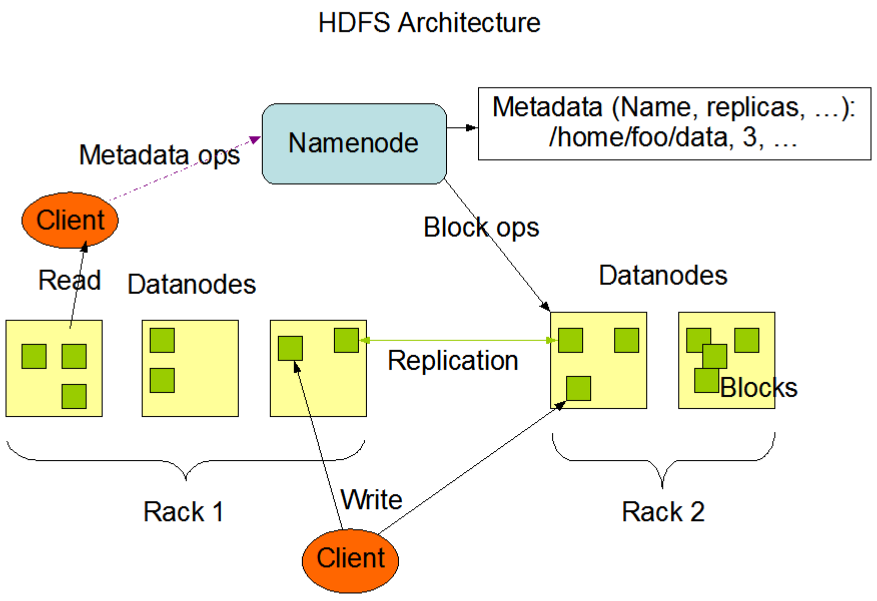


Figure 8.: HDFS Architecture (Source: `https://goo.gl/1zmtCq`)

A common operation such as reading a file or writing a file uses the NameNode to either know which DataNodes contain the block of the file or if it is possible to create a new File and which DataNodes to use to store. All of the actual reading and writing is made directly through the client and the DataNodes. The fact that the data is read directly from the DataNodes is what makes it very scalable.

A file in HDFS is divided in blocks, usually 64MB each which are spread among the several machines that belong to the system. The reason for such a large block lies in characteristic of traditional storage systems, Hard Disk Drives. They have a slow seek time compared to the transfer rate, thus by having large blocks , the seek time is reduced while taking advantage of transfer rate. Another reason for such large blocks is that each block has an overhead of the associated meta-data and if a large file is divided in small blocks then the overhead increases. The abstraction of a block not only allows to distribute a file through several machines but simplifies the storage system by reducing the amount of meta-data required by block science each block size is fixed and can be easily replicated, so if a block of a file is corrupted only that block must be replaced instead of the whole file.

A problem that hasn't been addressed is that the division of block is made by a fixed size and not logically. Thus if a file contains data in comma separated value (CSV) half of a line can be in a block and the other half in another. This is an issue that MapReduce works upon and is discussed in the next subsection.

2.4.2 *MapReduce*

MapReduce is a programming model and implementation based on functional programing that allows writing parallel code that can be executed in a large cluster without the developer having to know anything about distributed systems. This programing model divides the computation in two functions **map** and **Reduce**. The input data is divided by key and value that are fed to one or many parallel map functions. The map function thus has as input the key and value in which operates and returns a pair of key and value. The result of running a map over all the input data is joined and sorted according the keys thus gathering in a list all elements that have the same key. The final function, reduce, operates on top of key and list of values that resulted from joining. The result of reduce is another pair of key and value. This is the core of the computation and the user is only required to write the map and reduce function, since the function of joining is optional.

```
map :: k1 -> v1  -> (k2,v2)
join :: [(k2,v2)] -> [(k2,[v2])]
reduce :: k2 -> [v2] -> (k3,v3)
```

Suppose there is a file in HDFS that contains a new word in each line. To count how many alike there are a map function could take as key value pair, an empty key and the value of the word in each line. The map function would then emit as key the word and value the number 1. Afterwards the join function would join all the words with the same name and would create a list of ones for each word. The last function, reduce, simple has to go through the list and sum all the ones and obtain the total of words that are similar.

Algorithm 3 Map Word Count

```
Require: $Key, Value$
  return $(Value, 1)$
```

Algorithm 4 Reduce Word Count

```
Require: $Key$, $Values$
  $total \leftarrow 0$
  for $n$ in $Values$ do
    $total+ = n$
  end for
  return $(Key, total)$
```

In the concrete Hadoop implementaiton, a MapReduce job is composed of three tasks, map, join and reduce, which are controlled by the **TaskTracker**. Every machine in the cluster can have one TaskTracker that contains slots for tasks to be executed when commanded from the **JobTracker**. This last entity is responsible to accept and handle job submissions from clients, communicate with the NameNode to find where the input data is and coordinate with the TaskTrackers to complete the job.

As mentioned in the previous subsection the file is divided by blocks in a physical manner thus MapReduce needs some mechanist to be able to provide a logical key and value to the map function. The solution was to divide the original file in a logical fixed-size pieces called **splits** that are independent from the logical files. The splits are responsible for making a logical division and if a line for instance is divided in two HDFS blocks then both of these blocks will belong to the split. The split is then further divided into records that divided the split in key and values fed to the maps. An input split is made of its length in bytes and a set of storage locations that contain the logical data thus it is just a reference to the data.

2.4.3 *Hadoop Related Projects*

MapReduce is considered a low-level abstraction since the usual analysis task demand many MapReduce jobs. Due to the this and since many projects aim to take advantage of the Hadoop benefits and build on top of it. Apache **Pig** build upon Hadoop by providing a high-level language, **Pig Latin**, used to express data flows able to analyze the data. The programs written in Pig Latin are compiled to a sequence of MapReduce jobs to be executed. Pig was originally developed by Yahoo. Another similar project, Apache **Hive** simulates a traditional database on top of Hadoop by providing a SQL like query language, **HiveQL**. Hive allows the creation of tables such as a normal database and loads the data that is usually in text format to HDFS. Hive was created by Facebook when they started to explore Hadoop for their analyses needs and were able to have jobs that took a day to complete in their old system to be completed in a few hours with Hadoop (Thusoo et al., 2010).

HBase, modeled after BigTable (Chang et al., 2008), is a distributed key value store on top of HDFS but unlike previous projects does not use MapReduce. It it designed to provide real-time, random read/write access to the data unlike MapReduce. Even though it is not a RDBMS it still has the concept of table but there isn't a rigid schema since a row in a table can have different columns.

Tables rows are sorted by rows keys which are sorted in a byte-oriented fashion since keys and values are byte arrays. By having keys and values has byte array anything can be a key and anything can be stored. HBase automatically shards the data in *regions*. Initially a table only has one region that is divided in equal sized partitions has the size grows. Each new region is then attributed to a new node on the cluster. In the same line as MapReduce and HDFS, HBase uses a client/slave architecture. The master is responsible for managing regions and **regionservers**. The regionservers keep track of their own regions, attend requests from clients and the master. Additionally they are the ones responsible to create new splits and inform the master.

None of the existing projects provide a full fledged analytical system capable of answering complex SQL queries nor compute the entire data cube in a short time.To fill this gap the Apache project **Kylin**, still in the incubator provides a OLAP engine based on HBase and Hive. Kylin can be considered an HOLAP system, it works by storing the data in Hive tables and precomputes Cubes based on the meta-data of those tables and according to dimensions of interest. In case a certain query cannot be answered by a data-cube then it will be redirected to a hive query.

2.5 BENCHMARKS

Many different solutions have been developed by a diverse number of companies over the years for databases or OLAP servers but without a common ground it becomes difficult to compare which system is best, for what they are better suited, most importantly on how and why they behave differently. Benchmarks provide the grounds for comparison, by creating a common tool-set that can be used by anyone to calculate a relative performance usually used to answer the question of which system is best and at what tasks. While enterprises use benchmarks to showcase or improve their products, clients are able to use the results obtained to select the most adequate solution for their needs (Nambiar et al., 2009).

TPC is a non-profit organization formed by many members of respectable companies such as Oracle, IBM, Intel,etc, that creates and maintains several strands of benchmarks aimed at database applications. Aside from the provided benchmarks, enterprises that want to assess their products and publish them must write a report according to the guideline also set by TPC. Additionally the reports must be verified and certified by an independent auditor. Furthermore TPC is the first organization to require a price/performances comparison in all benchmarks (Nambiar et al., 2009).

Over time many benchmarks have been developed by TPC for different systems, some are now obsolete while others are being created. TPC-C (TPC, 2009) and TPC-E (TPC, 2014a) target OLTP databases by creating the database and loading it with data that can grow according to a parameterizable factor.

The database is evaluated with a workload simulating a complex OLTP environment through a mixture of read-only and update intensive transactions. The performance metric used in reports in both benchmarks is a “business throughput”. For TPC-C the main measure is transaction-per-minute-C (tmpC) while in TPC-E it is transactions-per-second-E (tpsE).

Decision support systems (DSS) such as OLAP servers can use TPC-H for benchmarking (Pöss and Floyd, 2000). Similar to the previous benchmarks TPC-H provides a data generator to create and load a 3rd normal schema where the entries of the table grow according to a scale factor. One of the main differences is the workload that consists of 22 complex ad-hoc, read-only queries that examine a large volume of data and answer real-world business questions (TPC, 2014b). The main performance metric used in this benchmark is TPC-H Composite Query-per-hour(QphH@Size) where size is the database size chosen for the measurement.

This measure manifests the system capability to process queries. As mentioned in the OLAP subsection current DSS do not store the information in a traditional 3rd normal schema but in a snowflake or star schema. Furthermore the number of tables and rows are much larger and most warehouses have to do maintenance operations. Due to the many problems (Nambiar and Poess, 2006) of TPC-H a new benchmark was created TCP-DH. This benchmarks focus on a snowflake schema, with 24 columns plus 18 columns on average. Additionally the workload is composed of a set of 99 SQL queries and 12 data maintenance operations. The performance metric used in TPC-DS, QphDS@SF, is similar to TPC-H but relies more on the scale factor instead of the database size.

TPC currently contains three more benchmarks, for energy consumption, virtual databases and Big Data (TPCx-HS), this last one even though it just sounds interesting it seems to just benchmarks HDFS.

2.6 SYNOPSIS

To provide a visual illustration of some topics reviewed on this section, a simple measure on how these topics are explored and how this dissertation relates with those topics, figure 9 depicts a concept-lattice that contains the relationship between the several topics and how many articles address such topics. The lattice was created using the tool Concept explorer (Yevtushenko, 2004) that has its basis in formal concept analysis. To understand the lattice it is enough to know that each reference was tagged with keywords that are the gray boxes and the number within the white boxes represent the number of articles with those keywords what percentage they represent on the total of all articles. For instance in the lattice it is possible to see that seven articles are related to benchmarks and they represent sixteen percent of all articles. Thus the circles with a blue top and black bottom represent the relation between keyword and quantities. The other circles with a white top and black bottom represent the quantities of article that have multiple keywords, for instance 2 articles focus on Data-Warehousing and OLAP. For this lattice few keywords that focus on the main topics were chosen, different keywords could have been used that related the articles in a different format. For instance no distinction could have been made between ROLAP and MOLAP and aggregate those articles in OLAP. But i choose to separate them so that OLAP represents general concepts while ROLAP and

MOLAP are a specific implementation. Nonetheless every article about Hadoop was tagged with the Keyword Big Data since there are no other solutions explored in this review. One important detail from looking at the lattice is that it seems that MOLAP has been more explored then ROLAP due to having a higher number of articles, but this is not true, the only motive for a higher number is that the ROLAP articles referred are big literature reviews themselves unlike the MOLAP articles that discuss separate topics. The focus of this dissertation is the implementation of a linear algebra approach to OLAP (LAOLAP) and as seen from the lattice it hasn't been deeply explored nor does it relate with any other topic.

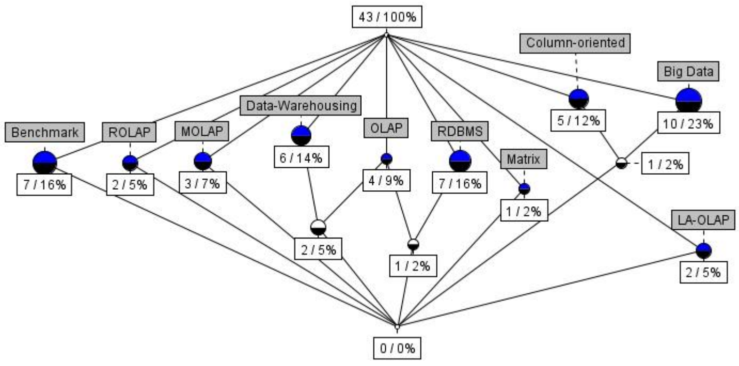


Figure 9.: Concept lattice of the most important topics

2.7 SUMMARY

This chapter summarizes the evolution of databases, from simple relation storage engines to powerful applications capable of handling the ever increasing size of information, while putting on the spotlight knowledge that will be useful throughout the dissertation. A brief introduction to the relational model is given which is essential to an understanding of some algorithms used in RDBMS. Then it quickly explains the need for OLAP engines, how data is commonly viewed (Data Cube) and then goes on presenting some algorithms for two different types of OLAP engines, OLAP and MOLAP. The topic of Big data is addressed next as well as how Hadoop is used in this context. This review would be incomplete without presenting tools commonly used to assess performance of different solutions, for instance the benchmarks provided by the TPC organization.

From all the topics presented we gather that even though there are well known solutions to OLAP systems but without any arithmetic properties to guide this systems. Furthermore with the new challenges of big data and due significant value of this systems we must rethink how these are developed. The benchmarks are used as beacon to these new solutions, by providing significant metrics on their performance.

3

THE PROBLEM AND ITS CHALLENGES

3.1 INTRODUCTION

This chapter follows a running example of how to calculate data aggregates in OLAP using the linear algebra encoding of data by Macedo and Oliveira (2014). This example serves to explain how the theory is applied to practical situations. Consider the following table recording the sales of a conjectured car shop. Sale records include car model, sale year, car color, quantity sold, month and season.

Model	Year	Color	Sales	Month	Seasson
Chevy	1990	Red	5	March	Spring
Chevy	1990	Blue	87	April	Spring
Ford	1990	Green	64	August	Summer
Ford	1990	Blue	99	October	Autumn
Ford	1991	Red	8	January	Winter
Ford	1991	Blue	7	January	Winter

Table 8.: Car sales

Suppose the shop owner needs to know which models sell better per season. This information can be straightforwardly obtained from the data using the SQL query language with the query 1.

Listing 1 Best selling model per season query.

```
SELECT Model, Season, sum (Sales)
FROM T
GROUP BY Model, Season
```

The result of the query is present at table 9.

(Chevy,Spring)	92
(Ford, Summer)	64
(Ford, Autumn)	99
(Ford, Winter)	15

Table 9.: Result of query 1

Thus this query yields nothing but a slice of the data cube with the features *Season* and *Model* having *Sales* as measure.

Following (Macedo and Oliveira, 2014), the previous query can be also described in linear algebra by the following matrix term:

$$t_{Model*Season} \cdot [\![T]\!]_{Sales} \cdot !^{\circ} \tag{1}$$

where

- $t_{Model*Season}$ — is the projection function associated to the product of dimensions *Model* and *Season*
- $[\![T]\!]_{Sales}$ — is the measure matrix (diagonal matrix representing the *Sales* column)
- $!^{\circ}$ — is the column vector wholly filled with 1s indicating that all data is to be *selected* in the consolidation.[1]

According to the theory behind this LA encoding of data,

$$t_{Model*Season} = t_{Model} \triangledown t_{Season}$$

where the joint projection is shown to be the same as the Khatri-Rao product ($\triangledown$) of the *projection functions* of each dimension attribute, in this case of *Model* and *Season*. The projection function t_A of an attribute A is a matrix that records the relation between the attribute values and the record numbers where these can be found. Such matrices represent functions because each record is bound to hold one and only one value per attribute.[2]

	0	1	2	3	4	5
Chevy	1	1	0	0	0	0
Ford	0	0	1	1	1	1

Table 10.: Projection function of Model ($_{tModel}$)

	0	1	2	3	4	5
Spring	1	1	0	0	0	0
Summer	0	0	1	0	0	0
Autumn	0	0	0	1	0	0
Winter	0	0	0	0	1	1

Table 11.: Projection function of season (t_{Season})

1 Row vector ! is usually referred to as the “bang” vector (Macedo and Oliveira, 2014).
2 These projections can be identified with the *bitmaps* of (Wu et al., 2006), regarded as matrices.

The standard definition of the Khatri-Rao product (KHP) requires two matrices with the same number of columns. For each column number, the corresponding column vectors are multiplied by the standard Kronecker product.

For instance, take column number 1 from both t_{Model} and t_{Season}. The corresponding column vectors are $[1, 0]$ and $[1, 0, 0, 0, 0]$, respectively written in transposed format to save space. The Kronecker product of such vectors will be (again transposed for space economy): $[1, 0, 0, 0, 0, 0, 0, 0]$, c.f. column 1 in Table 12.

In essence, this KHP operation simply computes the columns where the cross product of both attributes exist, for instance the attribute-pair $(Chevy, Spring)$ appears both in column 1 and 2 which are the lines 1 and 2 in the original table. Thus the first matrix of the formula 1 is calculated, $(t_{Model*Season})$. The matrices are presented with 0 based index similar to an array.

	0	1	2	3	4	5
(Chevy, Spring)	1	1	0	0	0	0
(Chevy, Summer)	0	0	0	0	0	0
(Chevy, Autumn)	0	0	0	0	0	0
(Chevy, Winter)	0	0	0	0	0	0
(Ford, Spring)	0	0	0	0	0	0
(Ford, Summer)	0	0	1	0	0	0
(Ford, Autumn)	0	0	0	1	0	0
(Ford, Winter)	0	0	0	0	1	1

Table 12.: Khatri-Rao product $t_{Model*Season} = t_{Model} \triangledown t_{Season}$

There are still two matrices in (1) calling for an explanation: the first is the *measure* matrix ($[\![T]\!]_{sales}$), a diagonal matrix keeping all the values of the $Sales$ column; the other is denoted by ! and is commonly referred to as the “bang” vector. Its converse is present in the formula ($!^{\circ}$) meaning that all records are to be taken into account. In general, this vector will be smaller than ! (but still Boolean) capturing a *selection predicate* typically implicit in SQL `where` clauses. In the example above all lines are selected and therefore the vector only contains 1s. But, should only sales over 15 be selected, this vector would have zeros in the first, fifth and sixth lines.

Once the projection function resulting from the Khatri-Rao product is multiplied by the measure matrix one obtains:

	0	1	2	3	4	5
(Chevy, Spring)	5	87	0	0	0	0
(Chevy, Summer)	0	0	0	0	0	0
(Chevy, Autumn)	0	0	0	0	0	0
(Chevy, Winter)	0	0	0	0	0	0
(Ford, Spring)	0	0	0	0	0	0
(Ford, Summer)	0	0	64	0	0	0
(Ford, Autumn)	0	0	0	99	0	0
(Ford, Winter)	0	0	0	0	8	7

Table 13.: Result of multiplying the *Sales* measure by $t_{Model*Season}$

1
1
1
1
1
1

Table 14.: Bang vector($!^{\circ}$)

Then the previous matrix and “bang converse” must be multiplied to obtain the final result:

(Chevy,Spring)	92
(Chevy,Summer)	0
(Chevy, Autumn)	0
(Chevy, Winter)	0
(Ford, Spring)	0
(Ford, Summer)	64
(Ford, Autumn)	99
(Ford, Winter)	15

Table 15.: Result of query 1 computed by LA

3.2 “DIVIDE AND CONQUER” STEP

Matrix composition (*aka* multiplication) in linear algebra can be performed using the following property,

$$\left[\, M \,\middle|\, N \,\right] \cdot \left[\begin{array}{c} P \\ \hline Q \end{array}\right] = M \cdot P + N \cdot Q \tag{2}$$

which allows us to divide matrices in two or more blocks, e.g. M and N in (2), to compose them separately and then to add up such intermediate results to obtain the final one. This is an instance of a generic, algorithmic process known as “divide and conquer” (Knuth, 1997/98). Formula (2) adopts the “block notation” proposed by (Macedo and Oliveira, 2014).

Bearing this principle in mind, our conjectured car-shop owner might have decided to buy two computers A and B and divide Table 8 horizontally in two halves, so that computer A holds the first three lines (records) of data and computer B holds the other three. Upon receiving their part of the data set, both machines will have to calculate the projection functions of the records they hold, as well as the corresponding measure matrices. As in (Macedo and Oliveira, 2014), it is assumed that each machine has global knowledge of every possible attribute value and its position (index) in the projection matrices. For instance, both know that $Summer$ is in the second position of the projection function of $Season$. Assuming this, the projection function for $Model$ and $Season$ and the Measure in each machine is as follows:

tModel			
Chevy	1	1	0
Ford	0	0	1

tSeason			
Spring	1	1	0
Summer	0	0	1
Autumn	0	0	0
Winter	0	0	0

Measure			
	5	0	0
	0	87	0
	0	0	64

Table 16.: Matrices in machine A

tModel			
Chevy	0	0	0
Ford	1	1	1

tSeason			
Spring	0	0	0
Summer	0	0	0
Autumn	1	0	0
Winter	0	1	1

Measure			
	99	0	0
	0	8	0
	0	0	7

Table 17.: Matrices in machine B

As in the previous section, each machine has to calculate the Khatri-Rao product of their projection matrices, leading to the following results:

	0	1	2
(Chevy, Spring)	1	1	0
(Chevy, Summer)	0	0	0
(Chevy, Autumn)	0	0	0
(Chevy, Winter)	0	0	0
(Ford, Spring)	0	0	0
(Ford, Summer)	0	0	1
(Ford, Autumn)	0	0	0
(Ford, Winter)	0	0	0

Table 18.: Khatri-Rao product of two projection matrices in machine A

	3	4	5
(Chevy, Spring)	0	0	0
(Chevy, Summer)	0	0	0
(Chevy, Autumn)	0	0	0
(Chevy, Winter)	0	0	0
(Ford, Spring)	0	0	0
(Ford, Summer)	0	0	0
(Ford, Autumn)	1	0	0
(Ford, Winter)	0	1	1

Table 19.: Khatri-Rao product of two projection matrices in machine B

To obtain the final result, two steps must still be accomplished which corresponds to the two LA formulas presented thus far. Firstly, the multiplication of the Khatri-Rao result by the measure matrix and then by the bang vector, as was done in the previous section. This multiplication of the three matrices corresponds to (1) restricted to the information that each machine holds. The same formula is also present in the multiplication of matrices in (2) where the $\left[\begin{array}{c|c} M & N \end{array}\right]$ corresponds to the Khatri-Rao product and $\left[\begin{array}{c} P \\ \hline Q \end{array}\right]$ is the result of the measure matrix multiplied by the bang vector.

Secondly, joining the results of each machine to obtain the final result corresponds to the sum of the last formula. With this, if $M \cdot P$ is the result in table 20 and $N \cdot Q$ is the result in table 21, the result of multiplying the three matrices in each machine and then summing them is present in the following tables:

(Chevy, Spring)	92
(Chevy, Summer)	0
(Chevy, Autumn)	0
(Chevy, Winter)	0
(Ford, Spring)	0
(Ford, Summer)	64
(Ford, Autumn)	0
(Ford, Winter)	0

Table 20.: Result vector in A

(Chevy, Spring)	0
(Chevy, Summer)	0
(Chevy, Autumn)	0
(Chevy, Winter)	0
(Ford, Spring)	0
(Ford, Summer)	0
(Ford, Autumn)	99
(Ford, Winter)	15

Table 21.: Result vector in B

(Chevy, Spring)	92
(Chevy, Summer)	0
(Chevy, Autumn)	0
(Chevy, Winter)	0
(Ford, Spring)	0
(Ford, Summer)	64
(Ford, Autumn)	99
(Ford, Winter)	15

Table 22.: Sum of the two vectors in machines A and B

The processes of joining both results is an instance of a *reduce step*. There is, however, a problem to address in the strategy presented above. Several solutions to such a problem are presented next and are further address in a chapter 7.

3.3 THE ATTRIBUTE RANGE PROBLEM.

Divide and conquer requires a strong assumption to be able to work in a distributed environment: each machine has somehow access to global knowledge of each attribute value position in the projection matrices. That is, when different entities create a projection matrix they do not assign the same attribute a different matrix row number. Should this not happen, the final sum of the results of each machine would not return a correct result.

Four solutions were considered to solve this problem: the first uses a central server, the second modifies the way the sum of the results is carried out by not needing the central server but leaves the realm of linear algebra; the third solution has to be further explored to see if it is possible to be implemented and consists on requesting the original database the index of the columns; the fourth uses a base 64 encoding to translate an attribute to a unique id and vice-versa.

CENTRAL COORDINATOR. A central server can easily solve this problem, by keeping a global mapping from attribute-values to row numbers, for every attribute (column in the raw data). Each machine could still parse and create a projection matrix individually, but when a new attribute was first encountered it would synchronize with the central coordinator. If that attribute already had a corresponding row number then the central server would return its line position otherwise it would simply assign a new line number and keep it stored.

Should the central coordinator be multi-threaded, the process of assigning a new row number has to be atomic so that no two different threads assign a different line number to an attribute. Each machine after synchronizing with the coordinator could keep a cache of the central coordinator.

(KEY,VALUE) SOLUTION. One can view the formula (2) in a different light, if we see the final result of each machine as key value pair where the key is the attributes cross product and value is the local result. When summing the results of the matrices instead of doing a simple matrix sum, it is feasible to join the results by its keys and sum the values in the case of Sum aggregation. Using table 20 and 21 if the key $(Ford, Spring)$ had value five in machine A and five in machine B the result of the sum could be achieved by summing the results obtained individually if the key was the same. This solution is similar to a MapReduce job.

COLUMN INDEX. If the original dataset resides in a OLTP database then it it might be possible to request an inverted index to the database which would contain a mapping from attribute to the line numbers where it appears. This solution has to be further analyzed since the index might point to memory address and not to line numbers, additionally not all columns are indexed and the original data source might not be a database, but the dump of one.

ATTRIBUTE ENCODING. A common process takes place when Hashing attributes, where an attribute can uniquely be hashed to a line number. Nonetheless hashing has collisions which can result in invalid results and hash functions are not generally reversible. However hash functions are meant to work with a finite number of addresses, if a matrix has as many rows as necessary, where the rows are not relevant contain only zero, then a solution can be obtained through Attribute encoding. By encoding the attributes by base 64 it is possible to obtain an unique id for each attribute in every machine and reverse the id to the original attribute.

3.4 SUMMARY

This chapter exposed the theory behind the overall strategy which the experiments must apply in a efficient way. The example given in the first section will be used throughout the dissertation to illustrate the application of linear algebra to the algorithms presented to solve some problems. From this small example we can divide the problem in several parts, the first being generating projection functions and measure matrices from the original data set. Once these are generated, the matrices must be stored in a compact format (since they are very sparse) and the mapping between matrix lines and attributes must be kept consistent.

Finally, in the “divide-and-conquer” strategy recommended for a distributed environment the original table must be divided in a set of chunks of data rows (as many as the processing units available) and the projection functions must be located in the same machine to reduce data transfer and keep the matrices consistent.

The following chapters will proceed to explaining two widespread sparse matrix storage formats, presenting their advantages and shortcomings and explaining candidate algorithms to solve such shortcomings. Afterwards different implementations are discussed depending on the solution for the attribute-range problem and how Hadoop might be used to help the development of the final application framework.

Part II

CORE OF THE DISSERTATION

4

STORAGE SOLUTIONS

4.1 INTRODUCTION

Matrix computations have been thoroughly researched in computer science due to their direct influence into the performance of scientific applications, computer graphics etc. This can also be observed in the Google page ranking algorithm, for instance.

The matrices used to encode the OLAP requirements will in general be very big and sparse. A projection matrix with n lines and m rows will have in the worst case a quadratic growth if n and m are the same, with a total of $n^2 - n$ zeros, since each column has at most one element. Therefore, they must be compressed to take advantage of their mathematical structure. For instance, the projection functions have very specific characteristics when regarded as Boolean matrices. Their converses, however, remain Boolean but become relations rather than functions. Considering these characteristics several improvements can be made on the matrix operations depending on the storage format. For instance, a correct Khatri-Rao product can easily be calculated without having to compute all the possible zero value results. Thus when possible the operations work with a compressed format in order not to waste time on decompressing them. This technique is similar to that used in column-store databases (Abadi et al., 2006, 2008). The storage formats that will be presented in the sequel are modified versions of the default formats *Compressed Sparse Column* (CSC) and *Coordinate Format* (CF) addressed by (Silva, 2005). In CSC columns are stored sequentially which as it be demonstrated are ideal for storing projection functions and Measures. On the other hand. the CF provides a simple solution to solve the attribute range problem within MapReduce when joining the results from different machines.

4.2 COMPRESSED SPARSE COLUMN

As the name suggests, this format compresses a given matrix by laying out the elements of a column sequentially in memory. In the standard format used by MKL four arrays are used. The first is the Values array, which contains the non-zero values from the matrix in a top-down, left to right fashion. The second array, Rows, contains the row number for each value. The last two arrays, pointerB ("B" stands for "begin") and ponterE ("E" stands for "end") indicate the values that belong to a row.

The following matrix will be used as an example to understand the MKL compressed format. It is just a simple example, where the sparsity is in fact very low and would occupy more space once compressed than decompressed.

$$\begin{array}{c|cccccc} & 0 & 1 & 2 & 3 & 4 & 5 \\ \hline 0 & 22 & 33 & 4 & 0 & 0 & 0 \\ 1 & 0 & 4 & 5 & 0 & 8 & 0 \\ 2 & 1 & 62 & 0 & 9 & 10 & 11 \\ 3 & 0 & 0 & 25 & 0 & 0 & 3 \end{array}$$

Figure 10.: Example Matrix

The following table contains the previous matrix in CSC format plus an additional index row to help visualizing the matrix:

Index	0	1	2	3	4	5	6	7	8	9	10	11	12
Values	22	1	33	4	62	4	5	25	9	8	10	11	3
Rows	0	2	0	1	2	0	1	3	2	1	2	2	3
PointerB	0	2	5	8	9	11							
PointerE	2	5	8	9	11	13							

Table 23.: Example matrix in CSC format

From the first position of *pointerB* its possible to know that the first column starts in the index 0 and from the *PointerE* that the first column ends in index 2. From this information it is possible to know that the first column has non-zero values 22 and 1, respectively in rows 0 and 2. It follows that rows 1 and 3 of this column are empty, i.e. hold 0. There is also a 3-array variation where PointerB and PointerE are grouped in just one array.

4.2.1 *Modified Compressed Sparse Column (MCSC)*

A possible modification to the CSC format can made which reduces size considerably thanks to a property of the specific matrices that are used. As pointed out before, projection matrices are functions, and therefore hold only one value (number 1) per column. Measure matrices hold at most one non-zero value per column.

Since these are the only types of matrices that are needed to calculate a Cube, both PointerB and PointerE can be removed. In the case of the projection functions the value array can also be removed due to all non-zero values being the same, number 1. In the case of measure matrices, each value is in the matrix diagonal. Therefore, its position in the column is also its position in the line. So, only the values array is needed in the case of measure matrices. If we are to use a matrix multiplication Library which requires matrices to be in the default format, the missing arrays can easily be created.

The following tables illustrate the compression of the $Sales$ measure matrix and of the projection function for dimension $Season$:

Values	5	87	64	99	8	7

Table 24.: Measure Matrix

Rows	0	0	1	2	3	3

Table 25.: Season projection function

There is, however, something missing from this example: the relation between row number and attribute value. This can be recorded by another array which contains the values of the attributes in their position, for instance: $Spring$ would reside in the first position of the array, $Summer$ in the second, $Autumn$ in the third and finally $Winter$ in the fourth position.

4.2.2 *Generating the projection function for MCSC*

Generating projection functions from raw data must ensure that (a) attribute values are in a one-to-one relationship with lines and that (b) their positions in the column must correspond to the same line number in the original table. Shall one these requirements be not fulfilled, the Khatri-Rao of two projection functions cannot ensure valid information. One could be tempted to sort the columns of the original table separately and then generating the projection function but this would not give a correct result after the Khatri-Rao product.

To solve these problems, an algorithm can either browse the original table row-wise and create the projection function of each dimension, or split the table in the different dimensions and generate each projection function. Assuming the second option and that the input is a list or an array of values ordered by their original positions it is relatively easy to generate the projection functions.

A possible algorithm for generating the projection function takes the array of values and has three variables, the result array, a mapping structure (variable map) that holds the attribute-line relationship, and the last line given to an attribute (variable Line) which starts at zero. For each element of the array, verify if its attribute already has a corresponding line in the map and if not then put a new entry in it where the key is the attribute and the value is the current number stored in the variable Line. Afterwards increment Line variable by 1. Additionally in each position of the attributes array it also writes the line number chosen to the resulting array. A possible implementation of the previous algorithm can be found in 7, that has the lines of the table starting in zero base index and creates a projection matrix in MCSC format.

4.2.3 *Calculating the Khatri-Rao product in MCSC*

The Khatri-Rao product can be calculated in the MCSC format without deserializing the matrix in a simple way since each column only has 1 value. Given matrix A of type $3 \leftarrow 5$ and matrix B of type $5 \leftarrow 5$, the resulting matrix $A \triangledown B = C$ will be of type $15 \leftarrow 5$.

The result of multiplying two lines with the Khatri-Rao product will fall in a position determined by the position of each line and the size of array B(K). Therefore the multiplication of every element in line m of matrix A and line n of matrix B will be in the position $k \times m + n$.

To calculate the Khatri-Rao the number of attributes of each matrix is required to calculate the length of the resulting array. With a simple loop from 0 to the length of the arrays, for each position simply apply the formula and put it in the result in the corresponding column since the column does not change, just the line number as can be seen in 8.

The next table exemplifies this algorithm for the function projection of season and model, where K equals to 4 since there are 4 lines in $Season$, M is the value of $Model$ at index X and N the value of $Season$ in index X.

Index	0	1	2	3	4	5
Model Row	0	0	1	1	1	1
Season Row	0	0	1	2	3	3
Result Row	0	0	5	6	7	7

Table 26.: Khatri-Rao product of tSeason and tModel

4.2.4 *Final result in MCSC*

To obtain the final result, there are two operation that need to be done: multiply the result of the Khatri-Rao product by the Measure and then by the selection vector, which in this case is “bang” ($!^{\circ}$). It is important to ensure that the index number of the Khatri-Rao product array correspond to the index number of the measure array, and that elements with the same row number sum together in the sum aggregation. Meaning that the value of the column at position N in the Khatri-Rao product array has its corresponding value at the measure array at position N. The presented operations take into account the special features of the matrices being multiplied (projections matrices are functions and measure matrices are diagonal). Therefore, matrices that do not exhibit the same characteristics can not be used in this multiplication operation. As such, the following algorithm is a special case of three matrix multiplication.

Once again the final result is easily obtained with a loop that iterates through the arrays length. The proposed algorithm takes as input: (a) the number of lines of the original projection functions; (b) the result of the Khatri-Rao; (c) the measure array; (d) the selection ($!^{\circ}$) array. First the result array is created with the size of multiplying the number of lines of both projections, every entry being zero.

The reason for this array size was already presented in the previous section. The next loop goes from 0 to the length of one of the input arrays. For position I of the loop, take the value of the selection array in position I in the variable B, the same for Measure but store it in the variable M, and again for the Khatri-Rao product and store it in K. Afterwards if B has value 1 then add M to the result array in position K. The Java snippet 9 contains the proposed algorithm.

The following table presents the final result, the missing corresponding records names can be obtained with a simple cross product, and the bang is omitted since it contains only 1 in the running example:

KH result	0	0	5	6	7	7		
Measure	5	87	64	99	8	7		
Final result	92	0	0	0	0	64	99	17

Table 27.: Result of the Final Multiplication

4.2.5 *Converse in MCSC*

An operator of Linear Algebra has yet not been discussed even though it is use in (Macedo and Oliveira, 2014) due to being a problematic operation in projection function. The converse of a projection function will represent the attributes in the columns and the lines number by the lines, making a column to possibly have more than one element which goes against the initial assumption. Using Season projection function in table 11 as an example the next table shows its converse and its representation in CSC.

Lines	*Spring*	*Summer*	*Autumn*	*Winter*
0	1	0	0	0
1	1	0	0	0
2	0	1	0	0
3	0	0	1	0
4	0	0	0	1
5	0	0	0	1

Table 28.: Converse of $tSeason$

Index	0	1	2	3	4	5
Values	1	1	1	1	1	1
Rows	0	1	2	3	4	5
PointerB	0	2	3	4		
PonterE	2	3	4	6		

Table 29.: $tSeason^{\circ}$ in CSC

No possible optimization can be made with this matrix aside from removing the Values array since it just contains the same value repeatedly and merging both pointer arrays.

To generate the converse of a MCSC matrix a simple algorithm can iterate through the Rows array and create an inverted index where the key is the line number contained in the array and the values are the column position of the line. Afterwards a simple iteration through the index can create the converse matrix in CSC. Table 30 represents the index created after browsing 25. This index can be used to create the converse matrix present in table 29.

Spring	0 1
Summer	2
Autumn	3
Winter	4 5

Table 30.: Converse index of $tSeason$

The operations presented in the previous sections, Khatri-Rao product and matrix multiplication which took advantage of the matrices properties can no longer be used if one of the matrix is conversed. The Khatri-Rao product can still be used because until now no where in LA theory there as been a Khatri-Rao product with a converse.

4.3 COORDINATE FORMAT

One of the most straightforward formats, Coordinate Format, stores only the non-zero values and their corresponding positions within the matrix. Additionally the values are not required to be in any order within the arrays. Unlike others formats, this only contains a three array variation, one to hold the values, another holding the row corresponding to the value and a last one regarding to the columns as illustrated in figure 31.

Index	0	1	2	3	4	5	6	7	8	9	10	11	12
Values	22	33	4	4	5	8	1	62	9	10	11	25	3
Rows	0	0	0	1	1	1	2	2	2	2	2	3	3
Columns	0	1	2	1	2	3	0	1	3	4	5	2	3

Table 31.: Example matrix 10 in CSC format

This format uses far more space than CSC and does not have any room from improvement without becoming identical to CSC. Nonetheless its simplicity allows us to easily do matrix multiplication of matrices with different origin tables on MapReduce by having the key as one coordinate while the value contains a pair with the other coordinate and the value. To demonstrate the application of CF within MapReduce let us suppose we have two projection functions t_a and t_b that reside in separate machines. In a cluster where the matrices have a consistent state, the operation $t_a \cdot t_b^\circ$ can be carried by running a similar but yet different Map depending on the projection function. If it is t_a then the keys of map output are the row numbers while the values are a tuple with columns and their corresponding entries On t_b the process is reversed, the keys are the rows and the values are a tuple of line number and the column number. Additionally each Map output value contains within the tuple the projection function of origin. Then the reduce part of the process simply has to do the multiplication and sum of the values depending on the origin of the tuple. This idea is illustrated in the following pseudo code.

Algorithm 5 Map Matrix Multiplication

```
Require: $t_a$
  for $row$ in $t_a$ do
    for $column$ in $t_a[row]$ do
      $value \leftarrow t_a[row][column]$
      $reduce \Leftarrow (row, (column, value, t_a))$
    end for
  end for
```

Algorithm 6 Reduce Matrix Multiplication

```
Require: $Key$, $Values$, $NumberOfLines$
  $result \leftarrow 0$
  for $i \leftarrow 0$ to $NumberOfLines$ do
    $val \leftarrow Values[t_a][i] * Values[t_b][i]$
    $result \leftarrow result + val$
  end for
  return $result$
```

4.4 SUMMARY

In this chapter two storage formats for sparse matrices were presented, CSC and CF. While in fact only one is actually used as a storage format, the others are also used to carry out specific operations. Both formats have a wide acceptance and have a long research history. As such, their typical usage in computer science was also addressed. Additionally, the CSC format had to be modified in order to use even less space. This was possible thanks to some properties inherent to the matrices used by the LA approach to data processing. Furthermore, it was shown how to carry out typical matrix operations without having to decompress this new format. Finally, an explanation was gven on the usefulness of the CF format in computing matrix multiplication over a MapReduce Cluster.

5 SYSTEM ARCHITECTURE

5.1 INTRODUCTION

Our benchmarks work on top of the Hadoop framework, as such we present in this section the several components of this system and how they are used. This framework was elected as it provides open-source technology with a distributed file system (HDFS) alongside a computational abstraction (MapReduce). The combination of these two technologies meets the requirements of having a cluster where each machine holds part of the total data and works as much as possible on its data part independently of the others. Moreover, some applications are built on top of this framework, with the goal of providing efficient data analysis. Thus there are several possibilities concerning the choice of a comparison engine, many of which are already used by industry (Facebook, HP and others are currently working to provide far more features). Note in passing that it is important to understand that while MapReduce might require a file system, HDFS can be used by other applications, independently from MapReduce. Such an example is a HBase that provides a Distributed Key-Value Store on top of HDFS.

The experiments built on top of this framework will have to deal with several limitations related to certain requirements from the LA theory for OLAP. The LA theory requires that a database table is partitioned horizontally, each part divided by the host machines, the generation of the matrices executed on each machine concurrently with the available data and that the matrix operations are also executed distributively according to their semantics. Alongside these objectives there is the requirement to keep every matrix consistent across the cluster. A problem that is not tackled in this work, is how to keep a set of related matrices replicated in the same machine so as to save transferring information across the cluster. Nonetheless, nearly identical issue has been address in (Floratou et al., 2011) by using a custom block management policy.

5.2 HADOOP FRAMEWORK

The Hadoop framework has four key modules, on its core there is the *Hadoop Common* that provides the abstraction from the underlying operating systems, the local file system and the scripts required to start the applications. On top of this module there is the Hadoop Distributed File System (HDFS) that abstracts the user from all the details of a file system divided in several hosts. With this application, it is possible to store large files and read them without noticing any difference from a local file system. HDFS by it self is already a very useful application, but it does not provide any way to carry out computation on the stored files without having to load them to the local machine and work on the entire file. To fill this gap we now have two different options, the first is the original MapReduce application and the other is an improvement on the previous version of MapReduce. The first application is a specific application of the MapReduce Model while the second iteration raises the level of abstraction and provides a more general framework capable of executing different computational models where the MapReduce is just a possible application.

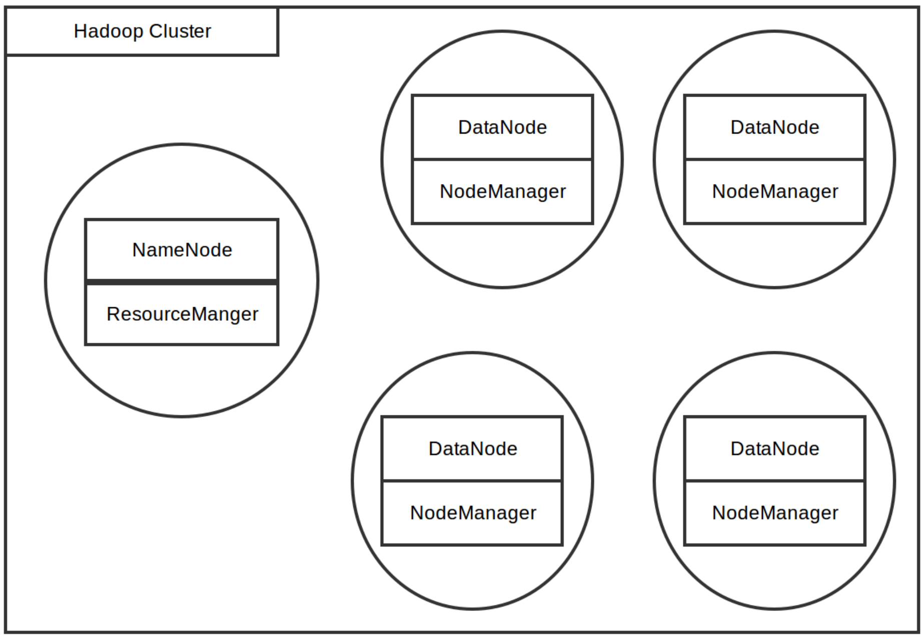


Figure 11.: Overview of theHadoop Cluster used in the Benchmarks

The cluster of the benchmarks uses two main components of the Hadoop Framework, the HDFS and the YARN Resource manager which is the improved version of the classic MapReduce. This selection was made because, not only are these two systems the current standard implementation of a Hadoop Cluster but because each one provides several key aspects for our solution. The Hadoop file systems by default already divides a file in several blocks and spreads this blocks evenly by the hosts. This aspect will facilitate the process of creating the matrices is a concurrent manner and divide their parts through the several computational hosts, as suggested by Macedo and Oliveira (2014). YARN and MapReduce provides a the abstraction to carry out the distributed computation of the LA theory and have a fine control on the resource usage of the cluster. The computation are carried out as jobs that the cluster must execute. Figure 11 depicts the system used to run the benchmarks, the Data Nodes control the blocks of the files stored in the HDFS while the Node Managers run the computing tasks. There are four servers with these daemons, there is another server that contains the entities that manage the entire cluster, the Name Node contains all the meta data of the HDFS and the Resource Manager that oversees the resources available in the cluster, how many jobs are being executed and the scheduling of this jobs. Some details on the HDFS are given in Subsection 2.4.1 and Section 3.3 addresses the implementation of matrix storage on the HDFS. Furthermore, Section 3.3 gives a deeper understanding of the difference between the implementations of the original MapReduce Application and The Yarn Application.

5.3 MATRIX GENERATION

The first step in order to generate the matrices, is to load in some way the data to the File System. TPC-H was elected as the benchmark to use in this research and on the generation of the data, several "tbl" files are created, one for each table. Thus the different files are loaded into the HDFS and by doing this they are automatically divided in blocks. Each host then holds a set of the blocks and these blocks are replicated amongst the machines according to certain properties that define the number of replicas and which policy to use when replicating. This division by blocks is done on a fixed size, for instance 64MB, thus it possible for a line of the "tbl" file to remain in different blocks.

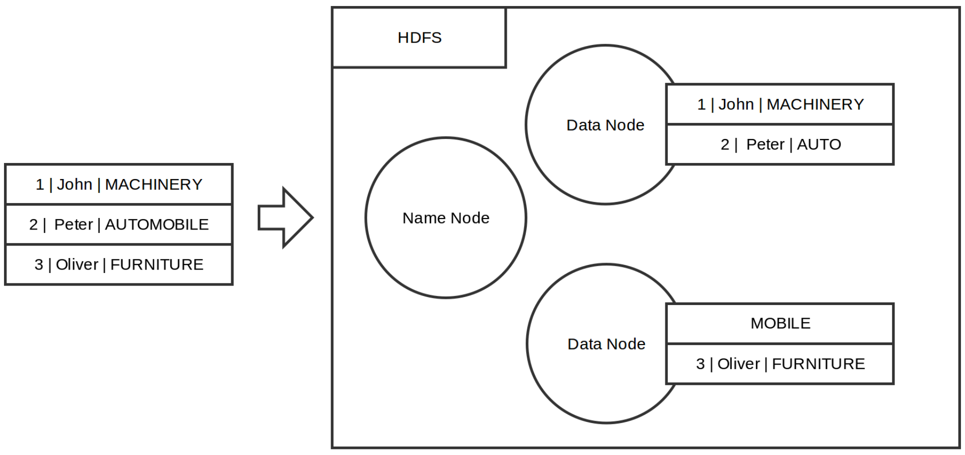


Figure 12.: Example of HDFS Block Partition

Following the storage of the matrices, comes the generation of matrices, but in order to accomplish this, there are three issues to tackle: if possible the generation of matrices has to be done where the block of files are stored: there has to be a coherent set of data to create a correct matrix and a unique line number for each attribute when the matrices are being generated concurrently. The first and second problem are tackled by Yarn and MapReduce, respectively. The other problem will be addressed in a following section.

5.3.1 *Yarn - The Resource Manager*

To deploy any MapReduce job on the cluster there must be a resource manager capable of executing a MapReduce request. Until the moment there are two iteration on this type of applications. The first, the classic MapReduce, is only capable of executing MapReduce jobs and is composed of two entities, the Job Tracker and the Task Traker. The first service is the central coordinate of the cluster, its responsibilities are to initializing a job, obtain the required input files, computing the input splits, divide the job in several task, assign those tasks to the taskTraker, oversee the job completion and update the client of the job progress. The TaskTrackers, one in each machine, usual in the same nodes of the HDFS data nodes, must execute the assigned tasks and trough a heartbeat, signal the Job Tracker of the task progress and possible errors. As expected the number of concurrent tasks is limited by the cluster resources, thus the task tracker must manage how many tasks can be executed at any given time. This management is done by slots, each task-tracker has a maximum number of slots for Maps and Reduces that are available. The amount of tasks can run in parallel in each is related to the number of processors available. Since the MapReduce is I/O Bound it is ideal to have more tasks then processors so that CPUS are used as much as possible. Thus if a machines has 4 CPUS, then 3 slots for Maps and 3 slots for Reduces is the recommended setting. The reason for being 3 slots instead of four is that the host usually contains the data node and task tracker which typical use as much resources as one slot. One JobTracker to oversee every job in the cluster makes this entity a single point of failure, furthermore the notion of slots to manage the cluster resource does not provide a fine grain control. Furthermore it has been noted that a cluster with more than 4000 nodes start to have scalability issues.

All of this issues are addressed by Yarn (Yet another resource manager). The JobTracker responsibilities are divided amongst the Resource Manager and the Application Manager. The Resource Manager tracks the cluster resources by having a deeper insight to the available resources, rather than keeping track of the available slots on each machine. In this solution each hosts dictates how much memory and virtual cores it has available at any given moment and tasks are assigned to these hosts depending on resource requirements of each task. For instance on task might require 2GB of main memory and 1 VCPU to run while another task might need 1GB of main memory but 4 VCPUS. The Application Manager, negotiates resources with the resource manager and schedules the tasks to be executed on the Cluster. Furthermore it keeps track of the running jobs and their overall state. There is another entity also added, the Application Master which oversees the flow of a single Job, thus for each MapReduce Job there is a dedicated Application Master. This separation of concerns turns the Map Reduce application an instance of an Application Manager, making it possible to have different types of applications running in the cluster. The task tracker is now called Node manager and handles the machine resources by providing the available number of virtual cups and memory available. With this new feature, a Map task is executed in a container that requires a certain number of CPUs and memory. With this minutiae in the system resources, the resource manager is capable of doing a fair use of the cluster and avoid under utilization of the total resources.

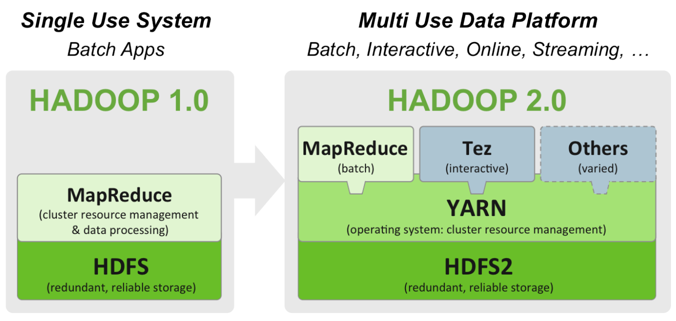

Single Use System
Batch Apps
Multi Use Data Platform
Batch, Interactive, Online, Streaming, ...
HADOOP 1.0
MapReduce
(cluster resource management & data processing)
HDFS
(redundant, reliable storage)
HADOOP 2.0
MapReduce
(batch)
Tez
(interactive)
Others
(varied)
YARN
(operating system: cluster resource management)
HDFS2
(redundant, reliable storage)

Figure 13.: Classic MapReduce and Yarn (Source: `http://goo.gl/FF3Yq0`)

5.3.2 *File Splits*

When the MapReduce application launches a Map container it must calculate the input split previously. An input split is a logic representation of the data since the blocks in HDFS are broken apart by a predetermined size. If the Map tasks worked directly with blocks then either some of the data would be corrupted or it would be developer job to retrieve the rest of the data residing in other HDFS block. To simplify this process, when a input split is calculated, depending on the file format that can be customized, the MapReduce client calculates how many blocks a split is composed of and stores that information. When calculating the file split, if one records starts in a HDFS block and ends in another then both blocks will compose the FileSplit. A input split might have a size bigger than a HDFS block, thus being composed of several blocks. When the tasks are assigned to a Node Manager the scheduler attempts to allocate the input splits based on their location, preferring to run task where the data resides, which reduces the overhead of data transfer. If it is not possible to allocate the jobs where the data resides then it prefers to assign to a machine in the same rack. This is one of fundamental principles, rack awareness, that make Hadoop so efficient, moving computation instead of data.

5.3.3 *Attribute Range Problem*

Two problems have already been solved by the Hadoop framework, partition the data horizontally in each machine and the matrices must be generated where the data resides. The next step is to generate the matrices and assign to each attribute a unique id. From all the options presented at Subsection 3.3 only two are used. The first option, a central coordinator, has the worse performance when compared to others, as is confirmed by testing the time it took to generate a small projection function far exceed the time it would take to generate the matrices with other options. Additionally by having a HBase in the same cluster, the resources that were available to execute MapReduce jobs diminished, since HBase uses a lot of memory and this would tend to increase has the attributes got bigger. Experiments were also made with a Nodejs application and a MySQL database but the time it took attribute a unique id to each attribute via a central coordinator is to large. After exploring the TPC-H Benchmarks it was clear that some columns do not have a fixed number of attributes such as dates while others do in fact have a small number of attributes. Also the data was generated to "tbl" files which meant that if it was desirable to use an index we would have to create one from the files. All of this factors aligned with the ongoing development of index in Hadoop applications such as Hive and and since not all columns in a database are indexed, option three, using database indexes to associate an attribute to an id, was discarded.

From all of the option only the following are left the Attribute encoding and the key value solution. These solutions provide a simple and fast mechanism to not only generate the matrices concurrently but also to decode and perform computation. The attribute encoding can be made in several ways, if the attribute is a String then it is encoded in 64 base that will return a number, this number will be unique for every attribute and will map to the line in the matrix. This number can also be latter decoded to obtain the corresponding attribute. If the attribute is a date or an Hour then this can be converted to their numerical representation in seconds, thus creating again a connection between the number and the attribute. There are some cases where no encoding is necessary, for instance when dealing with id column, which already contain unique numerical ids that serve as attribute and line number. All of this encoding mechanisms are used only in projection functions and are not required for Measures. Additional they create empty lines between attributes since some of the numbers generated although unique are not sequential, nonetheless this do not cause any problem if the matrices never have to be fully materialized and computation can be computed in a sparse format, otherwise it can lead to matrices that have many more elements then if the ids were generated sequentially. Table 32 contains a segment of table column and 33 the resulting MCSC by a process similar to the ones presented at subsection 4.2.1 but using attribute encoding. Note that since the attributes are encoded into row number there is no longer the need for a attribute array. In this example Peter is encoded to 25, Kyle to 100 and John to 100, as it can be noticed there would be several gaps if the matrix were directly decoded from this format.

Name	Peter	Kyle	John	Kyle	John	John

Table 32.: Database column with User names

Index	0	1	2	3	4	5
Rows	25	100	2	200	2	2

Table 33.: MCSC of the User names column

This solution is ideal when there are an infinite number of possible attributes and the matrix operation can be carried out without matrix materialization. This format is also very similar to a dictionary encoding used in column oriented databasesAbadi et al. (2008).

The Key value solution while not used for storage is used when after executing the matrix multiplication on the map side there is the need to aggregate results from different tables. It becomes natural to express in this case the matrix in a coordinate format where, for instance the key is the line number and the Value contains a tuple with both the column and the value in the matrix if there is any. An example of this usage has been portrayed in section 4.3

5.4 SUMMARY

Hadoop provides many of the features required to build a usable, generic Linear Algebra OLAP data analysis system by creating a custom Application Master, one of the core elements presented at this section. To have a working implementation several problems still have to be addressed that were not fully solved, such as a generic mechanism to distribute the matrices once stored or how to replicate the files in a congruent manner. The problems of distributing the data evenly among the data is solved by the MapReduce File Split abstraction and the requirement to have the matrices consistent across the cluster is solved using 64 base encoding and a coordinate matrix format. The issue of carrying out matrix computations concurrently in each machine is dictated by mathematical properties and not addressed here but gives rises to another path worth exploring of query execution plans for linear algebra equations.

6

TRANSLATING RELATION ALGEBRA TO LINEAR ALGEBRA

The typed LA approach to OLAP proposed by Macedo and Oliveira (2014) does not address how to translate relational algebra (RA) queries into linear algebra (LA) counterparts in a systematic and thorough way. When benchmarking this LA approach to data analytics we were faced with the fact that the analytical queries used in TPC-H are not just 'rollups' or 'cross tabs', but rather compute complex sub-cubes of the data, composed of several relational algebra operations.

By experience, from the analysis of some of the complex TPC-H OLAP queries it has become clear that the typed linear algebra must be extended to support all the operations commonly found in such queries. Clearly, it is necessary to have a complete transformation from relational algebra to linear algebra so that we may run any benchmark that uses standard SQL. Throughout this chapter we follow the definitions of relational operations proposed by Codd et al. (1993), that were later reviewed and extended by Pirotte (1982). As such, every relation is defined as a n-tuple where each element i as the Set S_i. This chapter will provide definitions for the most important operators and will show how to convert them to LA scripts.

6.1 PROJECTION

Data projection is an elementary operation both in relational algebra and in SQL, where it is incorporated in the "SELECT" statement. In very simple terms, a projection takes a relation, any number of attributes to be projected and creates another relation with the values of the initial relation but only for the projected attributes. Furthermore, it removes all duplicated tuples of the final relation. This last part is not the same in traditional databases, as they only remove duplicated tuples if such an instruction is given.

In the LA approach, attributes are divided into *dimensions* and *measures*, both represented by matrices: projection functions in the first case and diagonal matrices in the second. Where the typical case of a database would be to remove certain columns of a table, in the LA approach the process consists of joining the projection functions of the selected attributes. The process to achieve this must produce a final result with the same information as in a relational projection, ie. holding all possible combinations of the projected attributes and no duplicated tuples. As such, the easiest process to achieve this is by doing several Khatri-Rao products of the projection matrices. This poses a problem when we want to glue database columns which are seen as a measures. One way to solve this is by converting them into projection functions, but since the target of this LA encoding is to perform OLAP and this is not a typical use case, this situation will not be further addressed. Thus we assume that relational projections only involve dimension attributes.

Starting from the definition given by Codd (1970) for relational projection, let R be a n-ary relation and $K = \{A, B, ...\}$ be the set of its attributes. The L-projection of R, for $L \subseteq K$, is denoted by $\pi_L(R)$ and defined by $\pi_L(R) = R(A, B, \ldots, K)$ in standard relational algebra.

The corresponding LA encoding of this operation is obtained by the Khatri-Rao product of the projection functions corresponding to each attribute of the intended selection. Let r_X denote the projection matrix associated to attribute $X \in K$ in R. Then the LA outcome of projecting R by L is given by equation (3).

$$\pi_L(R) = \bigtriangledown_{X \in L} r_X \tag{3}$$

6.2 RESTRICTION

Restricting (or selecting) a relation R is the process of creating a sub-relation S that contains all the tuples of R that meet a certain condition. In the database language SQL this operation is embodied in the "WHERE" clause, which can be composed by several conditions. This section presents two formal definitions, one that restricts a relation on a single attribute and another definition that allows us to work with multiple restrictions.

In a formal definition along the lines of (Pirotte, 1982), a restriction on relation $R(K)$ with attributes in $|K|$ is the sub-relation $S = \sigma_\varphi(R)$ defined by

$$\sigma_\varphi(R) = \{t \in R \mid \varphi(t)\} \tag{4}$$

where φ is defined as $\varphi = X\,\theta\,c$ in which c is a constant of the domain D_X associated with X and θ is any binary comparison operation defined on that domain. The result S thus is a subset of R ($S \subseteq R$) in which $degree(S) = degree(R)$.

This definition can be further extended to have multiple restrictions on a single relation by composing them with binary logical operators (θ). For $\varphi = \alpha\theta\beta$, we have $\sigma_\varphi(R) = \sigma_\alpha(R)(\cap/\cup)\sigma_\beta(R)$ depending on the logical binary operator θ being $\vee$ or $\wedge$.

We start by addressing restrictions composed by the logical connector $\wedge$, leaving the logical connector $\vee$ for the end of this section. Below we present two methods to achieve a restriction on a single matrix. The first method works directly by transforming a projection function row-wise, turning entries to zeros wherever the attributes do not pass the given criteria.

6.2. Restriction

In what follows we use the notation of (Oliveira, 2012) for elements of a matrix M — $y\ M\ x$ indicates the value of the cell of M addressed by row y and column x. Let projection function $t_A : n \to |A|$ be given and $\varphi : |A| \to Bool$ be a restriction predicate on attribute A. Define $[\varphi] : 1 \to |A|$ as the column vector uniquely determined by φ — the fragment of $!^\circ : 1 \to |A|$ which indicates which values $a \in |A|$ satisfy φ:

$$a\ [\varphi]\_ = if\ \varphi(a)\ then\ 1\ else\ 0$$

Then we define the restriction of t_A by φ as follows:

$$\sigma_\varphi(t_A) = t_A \circ ([\varphi] \cdot !) \tag{5}$$

where $\circ$ denotes the Hadamard product (Million, 2007), i.e. the cell-wise multiplication between two matrices of the same type. Thus $! : n \to 1$ in (5). It is then possible to define that when φ is a predicate formula as the form $\varphi = \alpha \wedge \beta$, then $\sigma_\varphi(t_A)$ can be extended to $\sigma_\alpha(t_A) \circ \sigma_\beta(t_A)$.

Any form of restriction on projection functions according to this definition returns a matrix with the same dimensions and attributes as the arguments of the operations, the only difference being the values that the matrix holds. (That is, the type of $\sigma_\varphi(t_A)$ is always the same as that of $|A| \xleftarrow{t_A} n$.) Let us check this by calculating the pointwise meaning of (5):

$$\begin{array}{rcl} a(\sigma_\varphi(t_A))i & = & a(t_A \circ ([\varphi] \cdot !))i \\ & = & (a\ t_A\ i) \times (a\ ([\varphi] \cdot !)\ i) \\ & = & if\ a = t_A(i)\ then\ (a\ ([\varphi] \cdot !)\ i)\ else\ 0 \\ & = & if\ a = t_A(i) \wedge \varphi(a)\ then\ 1\ else\ 0 \end{array}$$

Consider as example the query 2 on table 8, which translates into LA script:

$$cs_{Model} \triangledown cs_{Year} \triangledown \sigma_{Color=Blue}(cs_{Color}) = cs_{Model} \triangledown cs_{Year} \triangledown cs_{Color} \circ ([Color = Blue] \cdot !) \tag{6}$$

This will produce the table found in 34.

Listing 2 Restriction Query

```
SELECT Model, Year, Color
FROM Car_sales as CS
WHERE CS.Color = Blue
```

A similar result can be obtained with equation (6) which uses the Khatri Rao product to calculate all the possible results and a restriction on the Color projection function. The result is illustrated in the table 35.

Model	Year	Color
Chevy	1990	Blue
Ford	1990	Blue
Ford	1991	Blue

Table 34.: Relational result of query 2

$$\begin{array}{r|cccccc}
 & 0 & 1 & 2 & 3 & 4 & 5 \\
\hline
Chevy \times 1990 \times Red & 0 & 0 & 0 & 0 & 0 & 0 \\
Chevy \times 1990 \times Blue & 1 & 0 & 0 & 0 & 0 & 0 \\
Chevy \times 1990 \times Green & 0 & 0 & 0 & 0 & 0 & 0 \\
\vdots & & & \vdots & & & \\
Ford \times 1990 \times Blue & 0 & 0 & 0 & 0 & 1 & 0 \\
\vdots & & & \vdots & & & \\
Ford \times 1991 \times Blue & 0 & 0 & 0 & 0 & 0 & 1
\end{array}$$

Table 35.: Result of equation (6)

However, the previously defined operation has one limitation: it becomes hard to express restrictions on an attribute that is not supposed to show up in the final result. One example of such a query can be found in listing 3.

Listing 3 Restriction Query

```
SELECT Model, Year
FROM Car_sales as CS
WHERE CS.Color = Blue
AND CS.Month = January
```

As anticipated earlier on, there is another way to carry out a restriction on a projection function that solves this problem. It involves constructing a different type of matrix, a binary diagonal matrix similar to a measure matrix that holds the values which are valid according to a certain criteria and then multiply the projection function by it. This matrix is defined in equation (7), where T is the original table, M an attribute of that table and φ is a predicate formula.

$$j[T]^{\varphi}_{M} i = \begin{cases} 1 & i = j \wedge \varphi(T(M,j)) \\ 0 & otherwise \end{cases} \tag{7}$$

Unlike the other restriction the matrix produced is of type $n \xleftarrow{[T]^{\varphi}_{M}} n$. In this way, a predicate formula composed by $\wedge$ can be achieved by simply generating as many matrices (7) as conjuncts and then mutiplying them using the Hadamard product, or simply chaining them by composition.[1] Thus we define the operation $\sigma_{\varphi}(t_A)$ for $\varphi = \alpha \wedge \beta$:

$$[T]^{\alpha \wedge \beta}_{M} = [T]^{\alpha}_{M} \cdot [T]^{\beta}_{M}$$

1 It is a well known result in relational and linear algebra that composition of Boolean diagonals is the same as their Hadamard product.

Following this approach, query 3 can be translated to the script (8). Its outcome in relational format is given by table 36. The corresponding matrix is given by 37.

$$(cs_{Model} \triangledown cs_{Year}) \cdot [CS]_{Color}^{Color=Blue} \cdot [CS]_{Month}^{Month=January} \tag{8}$$

Ford	1991

Table 36.: Relational result of query 3

$$\begin{array}{r|cccccc} & 0 & 1 & 2 & 3 & 4 & 5 \\ \hline Chevy \times 1990 & 0 & 0 & 0 & 0 & 0 & 0 \\ \vdots & & & \vdots & & & \\ Ford \times 1991 & 0 & 0 & 0 & 0 & 0 & 1 \end{array}$$

Table 37.: Result of equation (8)

Finally, we address selections in which $\varphi = \alpha \vee \beta$. Following Oliveira (2012) we define the following operation on Boolean diagonal matrices:

$$M \cup N = M + N - M \cdot N \tag{9}$$

This performs set union (predicate disjunction) on diagonal representation of sets (predicates).Then we define

$$[T]_M^{\alpha \vee \beta} = [T]_M^{\alpha} \cup [T]_M^{\beta}$$

6.3 JOINS

Joining tables is the most important operation on relational algebra as it provides a way to relate information that is spread out over different tables. To relate this information there must be a column on both tables with the same domain, so as to navigate between them. Relational databases provide many ways to execute a join; amongst them, there are natural joins, θ-joins, semi joins and so on. Even though the θ-joins may be a common operation, they pose a difficult task in converting to typed LA as they work on tables of the same type but different domains. When this is translated to LA, it means that the matrices will have a different number of rows, which makes it hard to operate on them. Thus this section will tackle the natural joins and semi joins, as they work on columns of the same domain. These joins already provide a solid foundation to create complex queries and develop further work on top of them.

6.3.1 *Natural join*

A natural join concatenates every tuple from a relation R with every tuple on a relation B through one or more attributes with common domains, as long as the values of such attributes are equal. Using as example tables 38 and 39, the natural join of both tables can be found in table 40. As can be seen by the examples, the resulting relation is the Cartesian product of all the tuples that have the same year.

Model	Year
Chevy	1990
Ford	1990
Ford	1991

Table 38.: Example Relation R

Year	Month	People
1990	June	10
1991	April	19
1991	July	20

Table 39.: Example relation S

Model	Year	Month	People
Chevy	1990	June	10
Ford	1990	June	10
Ford	1991	April	19
Ford	1991	July	20

Table 40.: Natural Join result of $R \bowtie S$

Let $R(L)$ be a relation with schema L and $t \in R$ be a tuple of R. For $X \subseteq L$, we denote by $t[X]$ the sub-tuple of t which contains only the values for the attributes in X. Following the more restrictive formal definition of (Pirotte, 1982), given two relations $R(L)$ and $S(K)$ such that some $A \in L$ and some $B \in K$ share the same domain of values, we define their natural join as the Cartesian product of both relations where condition $A = B$ holds. The formal definition of this operation is given in equation (10).

$$R \bowtie S = \{t1 \cup t2 \mid t1 \in R, t2 \in S, t1[A] = t2[B]\} \tag{10}$$

To convert the natural join to its linear algebra counterpart, the following steps have to be performed:

- From R generate projection matrices $r_A : n \to D$ and $r_{L-\{A\}} : n \to D'$ where n is the number of records in R and D and D' are the data domains of the corresponding attributes. **NB**: The iteration of the projection notation to a set of attributes is defined by Macedo and Oliveira (2014), that is:

$$r_L = \bigtriangledown_{X \in L} r_X \tag{11}$$

- From S generate projection matrices $s_B : m \to D$ and $s_{K-\{B\}} : m \to D''$ where m is the number of records in S. Note the same data domain D.

- Build

  $$R \bowtie S = (r_{L-\{A\}} \cdot r_A^\circ) \triangledown (s_{L-\{B\}} \cdot s_B^\circ) \tag{12}$$

  of type $D \to D' \times D''$. This formula is equivalent to

  $$R \bowtie S = (r_{L-\{A\}} \otimes s_{L-\{B\}}) \cdot (r_A^\circ \triangledown \cdot s_B^\circ)$$

  where $\otimes$ denotes matrix Kronecker product. This formula helps in grasping the meaning of a join in LA semantics, and how multiplicities are taken into account: $(d', d'')(R \bowtie S)d$ tells the number of occurrences of tuple $((d', d''), d)$ in $R \bowtie S$, calculated as follows:

  $$(d', d'')(R \bowtie S)d = \langle \Sigma i, j : d' = r_{L-\{A\}} i \wedge d'' = s_{L-\{B\}} j \wedge d = r_A i = s_B j : 1 \rangle$$

  where i and j are row indices.

Yet another formula for $R \bowtie S$ is given by

$$R \bowtie S = (r_{L-\{A\}} \triangledown id) \cdot r_A^\circ \cdot s_B \cdot (s_{L-\{B\}})^\circ \tag{13}$$

of the isomorphic type $D'' \to D' \times D$. (Note the term "$r_A^\circ \cdot s_B$, which is where the join effect" takes place.) All these alternatives contain the same information and are useful depending on "which attributes one wants on which side of the type arrow" ($\to$), in order to compose with the environment in which the join takes place.

Looking at these alternative definitions, equation (13) provides a clear separation that can be used to calculate the operation in a distributed environment. The multiplications regarding the projections matrices r can be computed in a set of machines while the the another set of machines computes the matrices on s. On each set, these computations can also be computed concurrently according to "divide and conquer" matrix multiplication, and by noting that $r_{L-\{A\}} = \bigtriangledown_{X \in L-\{A\}} r_X$ (11).

The step that forces the two sets of machines to synchronize is the "join term" $r_A^\circ \cdot s_B$. The join of the matrices on the left side of the join is calculated with a LA projection similar to the one presented above and multiplied by the matrix dot product of the matrix that joins the tables. This operation is given by the following equation $r_L \cdot r_A^\circ$ where the joining column is A and L are all the columns, A included. When applying this equation to the example of table 38, the result would be $(r_{Model} \triangledown r_{Year}) \cdot r_{Year}^\circ$ that is equivalent to $r_{Model} \triangledown img(r_{Year})$.

Since the Khatri-Rao product on projection functions is a commutative operation then it is possible to define it as $\bigtriangledown(t_L, img(t_C)) = \bigtriangledown_L t \triangledown img(t_C)^\circ$. This equation has the following type $\bigtriangledown(t_L, img(t_C)) : |L| \overset{t}{\longleftarrow} C$. Has can be seen by the type this operations relates every possible attribute combination with the column that is used to join the tables. This matrix has to be the last being multiplied and transposed so that the number of columns of the resulting matrix is equal to the number of rows to the matrix that will contain the right side of the join.

	June	April	July
$Chevy \times 1990$	10	0	0
$Chevy \times 1991$	0	0	0
$Ford \times 1990$	10	0	0
$Ford \times 1991$	0	19	20

Table 41.: Result of equation (15)

Now to tackle the right side of the join, it is already known that the resulting matrix must have on its rows the type B, has the natural join is made with a standard matrix multiplication. In the columns type there must be types of the columns of the Table S, which is obtained by another projection but with its result transposed. Additionally since the attributes are already on the Left side of the join multiplication then the right side does not need to include its type on the projection. Thus we arrive at the definition of the right side of the join, $s_B \cdot (s_{L-\{B\}})^\circ$. The right side of the joins has the type $s_B \cdot (s_{L-\{B\}})^\circ : B \xleftarrow{t} |L-B|$. Using the table 39 the resulting equation would be $s_{Year} \cdot s^\circ_{Month}$ if the People columns was not required, if it was, and since it a measure matrix then definition has to be slightly different has it would require a multiplication by the measure at the end, such as $s_{Year} \cdot (s_{Month} \cdot [\![s]\!]_{People})^\circ$. Thus the final formula for the right side is $t_C \cdot (\nabla^k_t \cdot \prod^M_{[\![t]\!]})^\circ$ where M are the measure matrices of the right side of the table. Subsequent to this addition the formula for the left side of join also must be update to $\prod^M_{[\![t]\!]} \cdot \nabla(t_K, img(t_C))$. The types of either formula does not change. After every definition, the transformation from a natural join to Linear algebra is given by equation (14) Where k are the required projection function regarding each table and M are the Measure matrices respectively.

$$R \bowtie S : |L| \leftarrow |K|$$
$$R \bowtie S = \prod_{[\![r]\!]}^{M} \cdot r_{L-\{A\}} \cdot r^\circ_A \cdot s_B \cdot (s_{L-\{B\}} \cdot \prod_{[\![s]\!]}^{M})^\circ \tag{14}$$

By applying this transformation to the tables 38 and 39 we get equation (15) that outputs the result in 41. As can be seen from these small examples, the same information is present in both algebras but represented in a different format.

$$Model \times Year \leftarrow Month$$
$$r_{Model} \triangledown r_{Year} \cdot r^\circ_{Year} \times s_{Year} \cdot (s_{Month} \cdot [\![s]\!]_{People})^\circ \tag{15}$$

Year	Month	People
1990	June	10
1992	April	19
1990	July	20

Table 42.: Updated example of relation B

Model	Year
Chevy	1990
Ford	1990

Table 43.: Result of the left side join of table 39 and 42

6.3.2 *Semi-Joins*

A *semi-join* is a one side natural join. What this means is that when joining two tables the end result is a subset of one of such tables where attributes match in a way defined below. The result will be taken from the left table of the join in case of the left semijoin ($\ltimes$) or from the right table in case of the right semi-join ($\rtimes$).

We give the definition for the left semi-join only, as the definition for the right semi-join is very similar. In relational algebra, the left semi-join of a relation R with S as in definition (10) is the sub relation $T \subseteq R$ which has all tuples in R for which there is a tuple in S that is equal on their common attribute names ($A = B$):

$$R \ltimes S = \{t1 \in R \mid \exists t2 \in S, t1[A] = t2[B]\} \tag{16}$$

Using as example table 42 (a different version version of table 39) the outcome of $A \ltimes B$ is given by table 43.

In order to translate this operator to LA we recall definition (13):

$$R \bowtie S = (r_{L-\{A\}} \triangledown id) \cdot r_A^\circ \cdot s_B \cdot (s_{L-\{B\}})^\circ$$

The case $R \ltimes S$ corresponds to the above once we ignore $s_{L-\{B\}}$, which can be done by replacing it by $! : m \to 1$:

$$R \ltimes S = (r_{L-\{A\}} \triangledown id) \cdot r_A^\circ \cdot s_B \cdot !^\circ \tag{17}$$

Note the type $1 \to D' \times D$, isomorphic to that of R.

There is another method to calculate a semi join, with different operations and a different purpose. The purpose of this method is to filter the right side matrix by the left side matrix so that the left side matrix type remains the same and is still a projection function. For this there will be a new operation similar to the matrix multiplication but instead of summing the products of multiplication it will multiply them. The definition for this operation is given in equation (18) where i is a row of r, j a

	0	1	2
1990	1	0	0
1991	0	0	0
1992	0	0	0

Table 44.: A_{Year} projection function left joined with B

column of s and k are the columns of r. Following the definition of this operation the position x_{00} of the resulting matrix is obtained by $r_{00} \cdot s_{00} \cdot r_{01} \cdot s_{10} \ldots r_{0k} \cdot s_{k0}$.

$$r \odot s = \prod_{k=1}^{m} r_{ik} \cdot s_{kj} \tag{18}$$

What this operation allows to do is to multiply any projection function by a vector and instead of summing how many elements there are for each attribute we simply get which attributes are present in that projection function. Now we can provide an equation that translate any semijoin into the linear algebra domain that produces a matrix that contains the same information as a relational join. Hence this translation is given in equation (19).

$$\begin{aligned} r \ltimes s &: |L| \leftarrow n \\ r \ltimes s &= (r^{\circ} \cdot (s \odot !^{\circ}))^{\circ} \end{aligned} \tag{19}$$

If the equation is not analyzed step by step it might seem convoluted. The transpose of A simply makes the attributes go to the top so that when the multiplication is carried out the types matches with the right side of the multiplication, hence the number of columns of A is equal to the number of rows of the product $(B \odot !^{\circ})$. The inner part of the equation filter the A matrix and puts zeros in the attribute lines that do not show up in the projection function B. The outer converse serves to return the projection function to its original format where the attributes are on the row. By applying the same semijoin as the one presented at the beginning of this subsection, the result of $A \ltimes B$ in Linear algebra is in table 44.

The right side semi join can is in Linear Algebra is nothing more than the left semi join but with the arguments switched as defined in equation 20.

$$r \rtimes s = s \ltimes r \tag{20}$$

6.3.3 *θ-Join*

Finally, concerning θ-joins we rely on the same definition (13), which we extend by inlining the Boolean matrix which encodes relational operator θ, for simplicity represented by the same symbol:

$$R \bowtie_{A\theta B} S = (r_{L-\{A\}} \triangledown id) \cdot (r_A^\circ \cdot \theta \cdot s_B) \cdot (s_{L-\{B\}})^\circ \tag{21}$$

This extends $R \bowtie S$, which corresponds to the special case $\theta = id$ representing $A = B$. Whith this we obtain a final equation (22) to carry out most join operations.

$$R \bowtie_\theta S : L \leftarrow K$$
$$R \bowtie_\theta S = \prod_{[\![r]\!]}^{M} \cdot r_{L-\{A\}} \cdot r_A^\circ \cdot \theta \cdot s_B \cdot (s_{L-\{B\}} \cdot \prod_{[\![s]\!]}^{M})^\circ \tag{22}$$

6.4 SUMMARY

This chapter contributes with some first steps required to enhance the application of typed LA approach to OLAP, making it usable to more complex queries similar to the ones used in OLAP benchmarking. In this sense it complements the LA encoding of `GROUP BY` queries given in chapter 3. Note however that not all relational operations (e.g. Antijoin and innerjoin). have been addressed.

Although far from a complete study, together with chapter 3 this chapter presents what may be regarded as the first steps towards a quantitative semantics of SQL, a kind of semantics required by its increasing use as a data analytical language.

7

BENCHMARKS

7.1 INTRODUCTION

This chapter is the apex of everything described thus far in this dissertation, from the sparse matrix formats to the SQL-LA conversion. It aims to assess whether the LA approach to data querying has potential or not to be an efficient solution in practice. To achieve this, we rely on the data formats described and apply the SQL-LA conversion rules to two derived queries from the TPC-H benchmark. This is followed by their implementation and execution on top of the Hadoop cluster, leading to an assessment of their performance both in terms of job latency and resource usage. Additionally, the performance of each query is compared to the results obtained in Hive, a system best suited to carry out the OLAP operations.

Throughout all the experiments the same system was used, under the same assumptions. One of the assumptions is that there are no failures on the system and as the experiments were built on top of the Hadoop framework this is not a unreasonable assumption, as the framework already deals with many failures. These include handling nodes which become unavailable, losing data either to corruption or a node failing; and also the failure of a task of a Job. Additionally, the scheduling of the resources in the cluster is not of special interest in these experiments as it is a controlled environment that runs only the tasks that we assign them. Furthermore, as the cluster has no multi tenancy we do not aim to evaluate the job throughput. As such and as mentioned above, we aim to just evaluate how long it takes to run each job, how large do the VM containers of the Hadoop tasks need to be and how many resources each node uses in terms of CPU, memory and disk.

All benchmarks were carried out in five machines, each with Ubuntu 14.04 64 bit running on Intel Core i3-3240 @ 3.40 Ghz, 3K cache and 8GB of RAM. One of these machines keeps the HDFS name nodes, the YARN resource manager and the Hive server. Every other machine contains both the HDFS data node and the YARN node manager. Each resource is given a 1GB JVM. In each machine, 4GB are made available to YARN which makes a cluster with a total amount of 16GB of RAM with 32 virtual cores. Each HDFS block has size 64MB.

Both approaches for every experiment distribute a database table horizontally through the nodes and both require a pre computation to convert the text files to their optimized format. In the experiments, Hive will be assessed with text file and optimized row format (ORC) without compression as our approach does not use any compression either. Every experiment consists of only one reduce step, over as many maps as data splits. The job latency was retrieved from the YARN job tracker that records several aspects of a job and the resource usage of every node was obtained using the Dstat tool.[1]

7.2 QUERY 1

For the first experiment we used a simpler version of TPC-H Query 1, as this query focuses on a single table, much like in the original paper Macedo and Oliveira (2014) and has a single filtering condition allowing us to to apply the translations defined in previous chapter, and it has multiple aggregations that could be computed independently.

However, as first exercise this query (listed as query 10 in appendix A) is too complex: it involves averaging, a type of consolidation not yet implemented in our experimental framework. So we start from something simpler and just calculate one aggregation without averages. We also simplify the schema of the table, given below for immediate and easy reference.

Listing 4 TPC-H Query 1 (simplified schema)

```
CREATE TABLE LINEITEM (
    QUANTITY    DECIMAL(15,2) NOT NULL,
    RETURNFLAG  CHAR(1) NOT NULL,
    LINESTATUS  CHAR(1) NOT NULL,
    SHIPDATE    DATE NOT NULL;
```

The query itself is given in listing 5. It involves a projection, an aggregation and a double-condition selection.

Listing 5 TPC-H Query 1 (simplified)

```
    SELECT RETURNFLAG, LINESTATUS, sum(QUANTITY)
    FROM LINEITEM
    WHERE SHIPDATE >= 1998-08-28 AND SHIPDATE <= 1998-12-01
    GROUP BY RETURNFLAG, LINESTATUS
```

The first step of the translation of this query to LA is to convert it into an relational algebra expression. For this, since "group-by" is not defined in standard relational algebra we assume a function SUM that, given a relation, aggregates the values over one of its columns. Then the query can be seen as a restriction on the relation $Lineitem$ (below abbreviated to L) in the column $Shipdate$ which is then projected on the columns $Returnflag$, $Linestatus$ and $Quantity$. This relational algebra expression of this SQL query is as follows:

1 http://dag.wiee.rs/home-made/dstat/

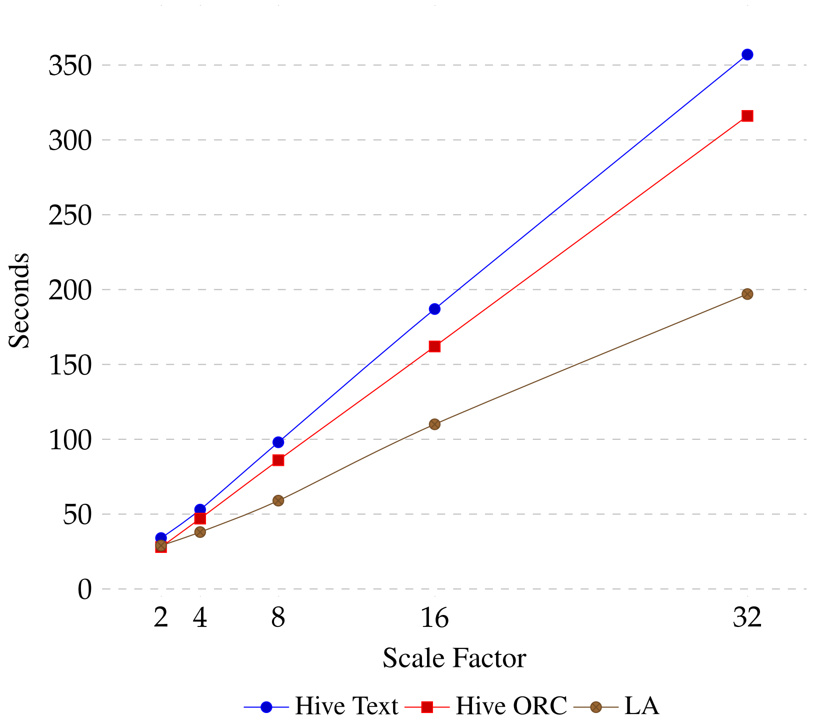


Figure 14.: Query1S Job Latency

$$\begin{aligned} &SUM_{Quantity}(\pi_{(ReturnFlag,LineStatus,Quantity)}(restriction)\ \textbf{where} \\ &restriction = \sigma_{Shipdate>=1998-08-28 \wedge Shipdate<=1998-12-01}(L) \end{aligned} \tag{23}$$

By applying the definitions presented in chapter 6 and knowing that a group-by is achieved in LA by the Khatri Rao and a bang vector to aggregate, we get LA script (24) that neatly expresses the pipeline of the operations involved.

$$\underbrace{(L_{ReturnFlag} \triangledown L_{LineStatus})}_{\text{projection}} \cdot \underbrace{[L]^{Shipdate>=1998-08-28}_{Shipdate} \cdot [L]^{Shipdate<=1998-12-01}_{Shipdate}}_{\text{selection}} \cdot \underbrace{[\![L]\!]_{Quantity} \cdot !^{\circ}}_{\text{aggregation}} \tag{24}$$

The query execution plan for both approaches are similar: most of the computation is carried out in the map phase, whereupon values are aggregated in the reduce phase. In the linear algebra case, the map reads the matrix lazily, filters the elements according to the condition and sends them to the reduce step to be aggregated. The experiment was executed with data generated by the TPC-H *LineItem* table with different scale factors generating tables of an increasing size. The scale factor 2 generates a table with an approximate size of 1.5GB while scale factor 32 yields a table with size 23.50GB.

Figure 14 presents the average time it took to complete a job in the cluster. As can be observed, our approach exhibits an improvement on the time it takes to compute the results. This is expected, as our approach has the columns of the table divided and only needs to read the projection functions that contain the data of those columns — while the other approaches must read files that contain extra information. These results are in line with the ones found at Floratou et al. (2011) which also presents and explains some of the problems we had in the Hadoop implementation.

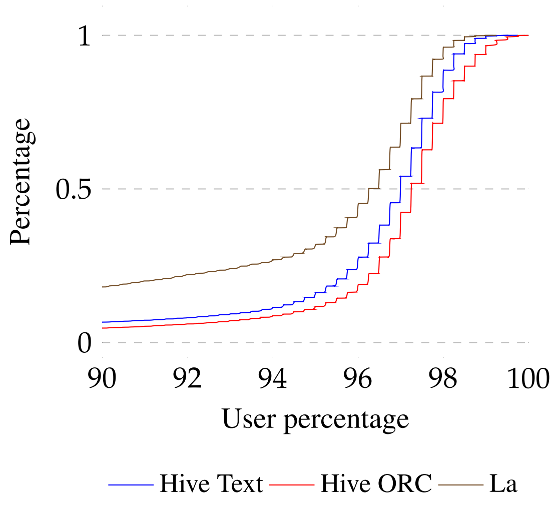


Figure 15.: CPU User

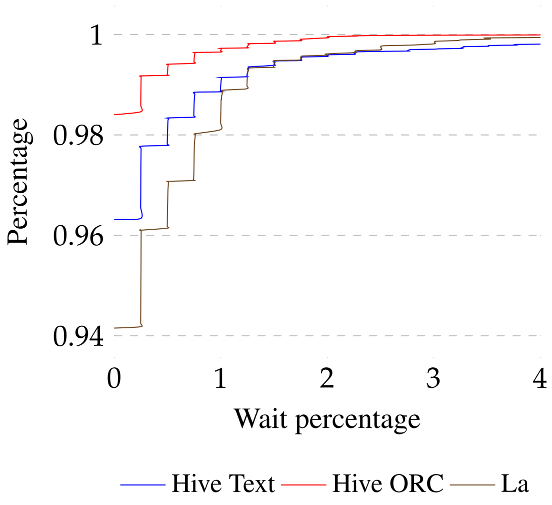


Figure 16.: IO wait

The results presented in Figure 15 and 16 are a CDF (cumulative Distribution function) from all the experiments of the CPU usage using the "dstat" tool on the nodes that carries out the computation. From these plots we gather that the ORC format does a much better job at using the CPU as its usage percentage is between 96 and 100 while spending less time waiting for I/O operations. On the other hand our approach has the CPU usage more distributed between 90 and 100 which means that it spends more time waiting on I/O operations as can be seen in image 16. In the same plot it cant be perceived but there is a slight difference on the I/O time from the LA approach and the ORC has the first never has waiting percentage of 7% while the max waiting percentage of La is 30%; meaning that in none of the approaches the wait time is worrisome but the ORC format spends considerably less time on I/O wait.

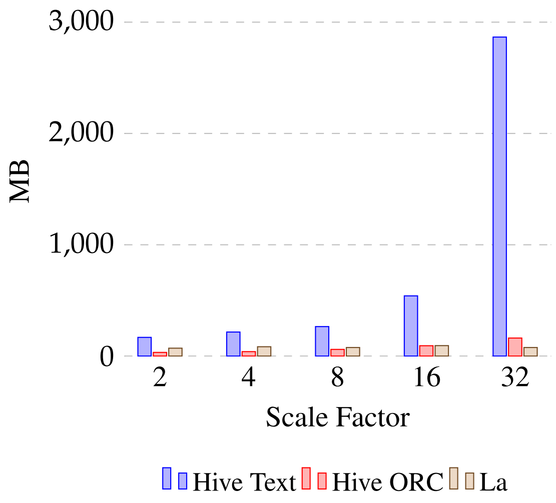


Figure 17.: Data read

As Hadoop either reads a block of a file locally if available or reads it through the network when not, we decided to aggregate the values from both channels to see which approach needs to read the least amount of bytes. From figure 16 it becomes clear that the textual format used in Hive is the least efficient while a distinct pattern can't be found in the other approaches. Nonetheless the Hive ORC on average seems to use less data on smaller sizes while our approach seems to use less data as scale factors increases. These results relate nicely with the latency time, which explains why our approach terminates much faster: it needs to read far less information.

7.3 QUERY 3

For the last experiment we have a more complex query involving a table join, which is a simplified version of TPCH-H query3 (cf. listing 11). Unlike the previous query, this one works on multiple tables and aggregates data from both of them with a single join. The original TPC-H Query3, on the other hand, only aggregates data from a single table but has multiple joins to filter the results and some selections on different tables, a pattern already addressed in the first query. Furthermore, the original query has an `ORDER BY` operation that is not relevant in the current setting. (This does not seem a complex operation; it can be easily added in the future as it seems at first sight a simple permutation on the order of the matrix rows in the final result.) The query that will be evaluated in this section is presented in listing 6.

By following the same structure and conventions as in the previous sections, first we transform the SQL query 6 to a relational algebra expression and then to a typed linear algebra equation. In this case we will suppose that it exists a function SUM that takes a relation and aggregates the tuples on the columns $shipmode$ and $orderstatus$ and returns a new relation that joins the values of the Sets $totalprice$ and $quantity$ according to the expression $totalprice * quantity$. Then the query is simply

Listing 6 TPC-H Query 3 (simplified)

```
SELECT  L_SHIPMODE,
        O_ORDERSTATUS,
        SUM(O_TOTALPRICE * L_QUANTITY)
FROM LINEITEM AS L, ORDERS AS O
WHERE L_ORDERKEY = O_ORDERKEY
GROUP BY L_SHIPMODE, O_ORDERSTATUS.
```

the application of the function SUM to the result of natural join between two projected relations. On the left side the relation line item is projected on $shipmode$, $quantity$ and $orderkey$ while on the right side the relation $orders$ is projected on $orderstatus$, $totalprice$ and $orderkey$. This expression is found in equation 25.

$$\begin{aligned} pLineItem &= \pi_{(shipmode,quantity,orderkey)}(LineItem) \\ pOrders &= \pi_{(orderstatus,totalprice,orderkey)}(Orders) \\ & SUM(pLineItem \bowtie_{orderkey} pOrders) \end{aligned} \tag{25}$$

Using the equations presented to go from relational algebra to typed linear algebra the result of this transformation is equation 26. To understand the resulting equation one must not forget that the projection on LA work only on projection functions and even thought not explained in the chapter 6 the converse of square diagonal matrix M is M itself, meaning $M^{\circ} = M$.

$$(L_{shipmode} \cdot [\![L]\!]_{quantity} \cdot L^{\circ}_{orderkey}) \cdot (O_{orderkey} \cdot [\![O]\!]_{totalprice} \cdot O_{orderstatus}) \tag{26}$$

Similar to the previous query, the execution plan for both approaches are similar, the query is divided in two Stages, the first stage divides the computation on the two sides of the join. In each side it is calculated concurrently the projections and aggregated the values of each tables. The last phase is the final aggregation of all the results and creation of the result. Each phase is also further divided into as many map tasks as required and a single reduce to keep similar across all the experiments. The line item table is also the same as the one used in the previous benchmark. The other table, Orders, has on the scale factor 2 an approximate size of 330MB and on the scale factor 32 a total amount of 5.3GB.

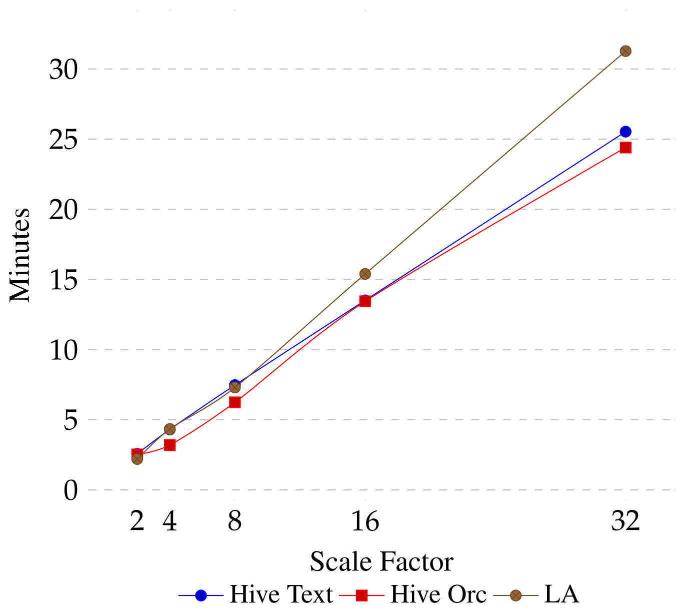


Figure 18.: Query3S Job Latency aggregated Stages

Figure 18 contains the average total amount of time it takes to complete a job with both stages for each scale factor. One thing that stands out is the average time it takes to complete the queries is much higher than the previous queries which shows the increased complexity of the query. Unlike the other query our approach is not as efficient as any of the other approach. I attribute this decrease in efficiency to the fact that the matrix multiplications can not be computed in a lazy manner as in the Khatri Rao. This leads to the fact that each multiplication must be computed in memory, which requires more memory for each map task. While our approach requires at most a total of 2GB of main memory the other approaches only use at most 1GB. This means that our approach can only have at most 12 map tasks running concurrently on the cluster while the other approaches can have up to 24.

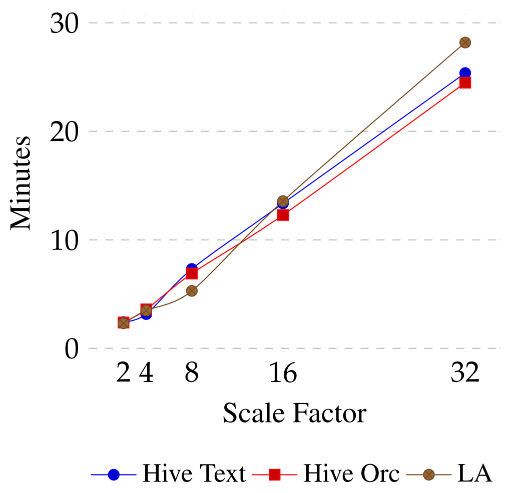


Figure 19.: Latency Stage 1

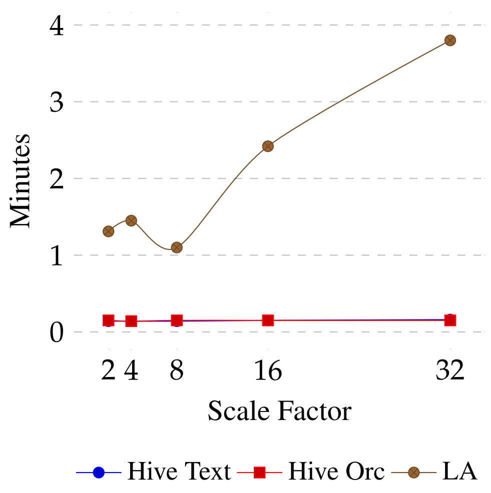


Figure 20.: Latency Stage2

The Figures 19 and 20 show the jobs latency divided by stage one and stage two respectively. From this plots it can be derived that most of the computational effort in either approach is on the first stage which corresponds with the expectations as in both of them is when the original data is read and aggregated as much as possible. The final Stage on both hive formats does not have a high impact on the overall latency time, but in the case of the LA approach the fact that it has to multiply two big matrices causes the overall time to have a slight increase. At the early scale factors of Stage 1 our approach has a slight advantage, maybe due to the fact that the the number of concurrent map tasks in the cluster is small and is only reading the required columns for each matrix.

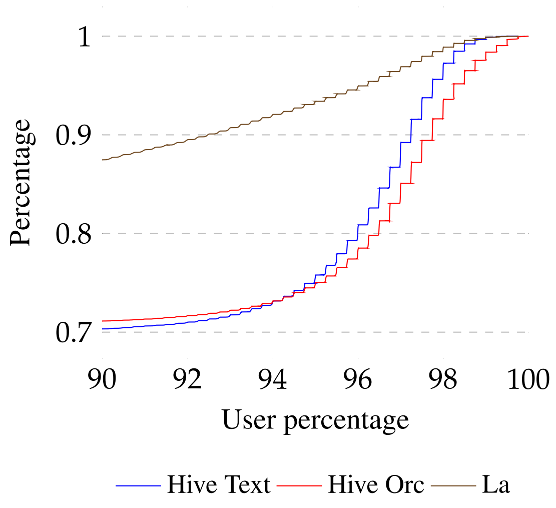


Figure 21.: CPU User

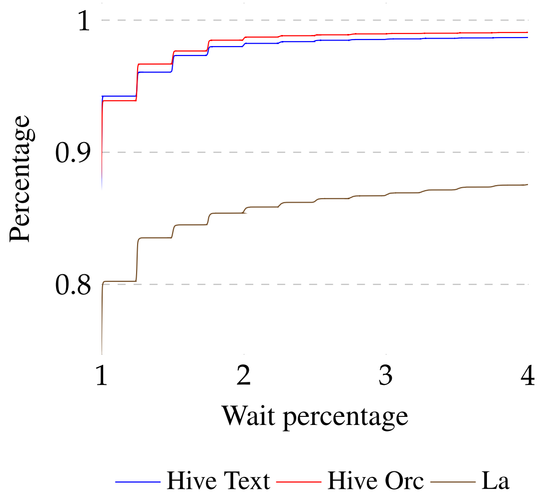


Figure 22.: IO wait

From the Figures 21 and 22 that have the CPU usage of the Query with all the stages in the same way that was presented in the previous benchmark. And much like in the previous benchmark the Hive approaches have a better CPU utilization. In this case the typed linear algebra approach actually spends more time in I/O wait than the previous benchmark.

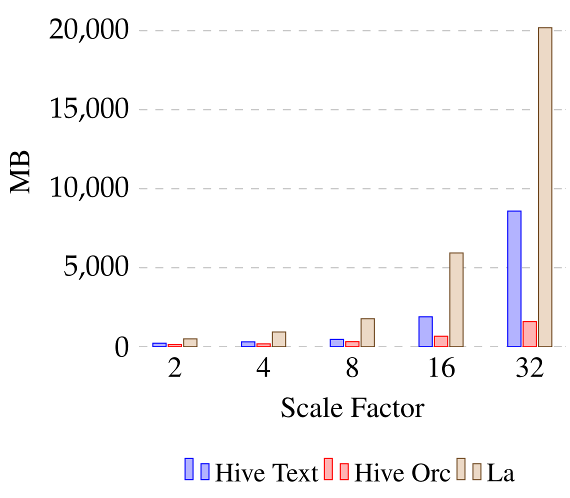


Figure 23.: Data read

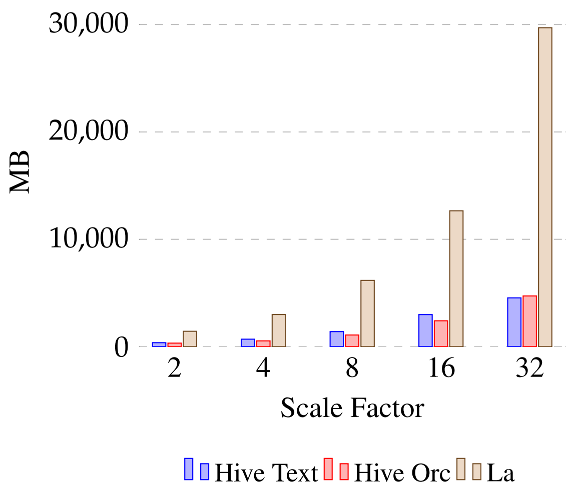


Figure 24.: Data written

Once more, similar to what was done in the previous section the amount of data that was read on each machine was aggregated on by what was read from disk and network. In this case we also did the same process for the quantity of data that was written. Figure 23 contains the amount of data that was read for each scale factor while 24 has the amount written for each scale factor. It is worth mentioning that this is the total amount from both Stage one and two on both plots. Our Linear Algebra approach read far more information than the both Hive approaches which goes against our expectation that by dividing by matrices we read far less data. Thus we also analyzed the total amount of data that each approach writes. This information correlates with the CPU usage plots 21 and 22 where the Linear Algebra approach spends a reasonable amount of time with I/O. However this results are not similar to the benchmark Query1 has they are aggregated of two stages where each one has several maps and one reduce. From these results Hive is capable of aggregating much more information on the map tasks and the on the first stage which also explains why its second stage has an almost constant time over the several factor. Moreover has the multiplication on both sides of the join form a dense matrix with a far bigger sizes that must also be multiplied and form another one make the total amount of data being read by our approach significantly higher.

7.4 SUMMARY

Three approaches to two benchmarks have been evaluated, each one having a different storage format and behaving with slight differences. These experiments provide not only an insight into the performance of a Linear Algebra approach to OLAP but also into the performance of a state of the art application. Each query had a special purpose, focused on tackling a particular aspect of the LA implementation. The first query evaluates the computation of a sub-cube made of a single table, as handled in (Macedo and Oliveira, 2014) with the addition of some constraints. The second query focused on the computation of a sub-cube also, but composed of several tables that were joined by columns of the same domain. Furthermore, for each query we applied the strategy developed in chapter 6 for translating SQL queries to LA scripts. For each trial, the latency of the job execution time was evaluated together with several statistics on the computational requirements in terms of resources.

Altogether, from the experiments with the first query we conclude that a lazy computation of matrices (which is similar to the processing of columns) seems to have clear advantage over the Hive implementation by having a lower latency time and by reading much less information. On the other hand, query 3 presents a challenge to our approach as we witness a slight decrease in performance due to creation of dense matrices that require far more resources in terms of the amount of data written / read, which in turn lead to the CPU spending more time in I/O wait.

8 CONCLUSION

8.1 CONCLUSIONS

A novel approach to the computation of OLAP queries (Macedo and Oliveira, 2014) has opened up a new research niche with many open questions on both the formalization of the quantitative side of OLAP operations using LA, and on efficient (typed) sparse matrix representation required by such an approach. This work was set up with the main goal of tackling these questions and benchmarking the performance of this approach in contrast to the existing, standard solutions.

This dissertation gives an account of the ingredients of the overall approach and its benchmarking, starting by providing a simple state-of-the-art understanding of database systems, their recent evolution towards big data, and some of the existing technologies shaping up for the future. This is followed by introducing the main ideas presented by Macedo and Oliveira (2014) and anticipating potential difficulties that a possible implementation of this algebraic approach might face up in a distributed setting. This includes very big and sparse matrix storage solutions and a possible system architecture for a distributed application. Furthermore, efforts have also been made towards creating a consistent transformation scheme from relational algebra to typed linear algebra so that LA scripts can be automatically generated from standard SQL queries. Finally, the last chapter of the core of the dissertation sets up a number of experiments which put to trial the solutions developed beforehand by assessing their performance.

At the beginning of this research we expected that matrix computations would be performed on top of very efficient, off-the-shelf low level libraries. However, such libraries did not meet such expectations as, for instance, most of them do not handle sparse matrices as we had expected; operations on sparse matrices always return dense matrices; and some operations (such as Khatri-Rao matrix multiplication) are not provided. For these reasons we implemented a custom format and the operations that were required for the LA encoding and experimentation.

To keep our experiments close to the goals of the original paper by Macedo and Oliveira (2014), the computations were carried out in a distributed environment with the help of the Hadoop framework. In this implementation we also met some adversities on keeping the data distributed horizontally through the nodes with replication. However, since this problem was not among the initial requirements and has been studied elsewhere through the usage of replication policies, we did not give a high relevance to this topic. The transformation between SQL and LA still leaves many doors open to explore, even though it provides a first, significant step.

8.2 FUTURE WORK

The benchmarks presented in this dissertation provide a first assessment of an LA approach to data analytics. However, this assessment is not thorough enough in the sense that many other LA scripts should be put to test, covering more and more complex situations in data analytics. The main reason for not having done such experiments thus far has to do with the rather laborious process of creating them, which is far from being automatic. As such, we regard the implementation of a complete set of rules for encoding SQL queries into LA scripts as a first and foremost need.

Besides its practical usefulness for benchmarking, such a set of rules will open other branches of this research. First, one can start studying the optimization of the SQL-LA transformation in a distributed data setting. Second, and independently of any performance gains on the implementation, the transformation will provide a complete formalization and verification of OLAP querying in the rich mathematics of linear algebra. Thus verification becomes possible and OLAP databases will be provided with the capability to ensure the data analyst that the operations taking place are correct. Third, the LA background of (Macedo and Oliveira, 2014) will be challenged with the likely need to provide new LA combinators able to cover the real needs of a practical implementation of an OLAP system where datasets become matrices.

Armed with all this knowledge, it will be easier to fine tune our research in the area of efficient matrix representation. Indeed, this novel application of linear algebra can help this field by adding new operations and/on new formats that handle particularly sparse matrix multiplications efficiently. In fact, there is a well organized taxonomy of binary relations (Oliveira, 2008) which will surely reflect on a similar taxonomy of (sparse) matrices, as first steps in this direction already indicate (Desharnais et al., 2014).

Once such formats are (hopefully) developed and efficient matrix computations are implemented for them, a well structured repository of experiments can be set up covering the full spectrum of the TPC-H benchmarks, giving a deeper understanding of where the LA theory has advantages or disadvantages in terms of resource usage, and finally applying the benchmarks in a system with several concurrent users.

Even if matrix multiplication remains not as efficient as expected we can always implement the approach in a *columnar* setting (Floratou et al., 2011), which our first benchmark in a sense suggests as a viable alternative.

Summing up, we suggest for future work the following broad tasks: complete the SQL to LA encoding, implement a corresponding low level matrix library, develop a real, fully operational implementation and finally carry out a thorough benchmark of the system.

BIBLIOGRAPHY

D. J. Abadi, S. Madden, and M. Ferreira. Integrating compression and execution in column-oriented database systems. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, Chicago, Illinois, USA, June 27-29, 2006*, pages 671–682, 2006. doi: 10.1145/1142473.1142548. URL http://doi.acm.org/10.1145/1142473.1142548.

D.J. Abadi, S. Madden, and N. Hachem. Column-stores vs. row-stores: how different are they really? In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2008, Vancouver, BC, Canada, June 10-12, 2008*, pages 967–980, 2008. doi: 10.1145/1376616.1376712. URL http://doi.acm.org/10.1145/1376616.1376712.

S. Agarwal, R. Agrawal, P. Deshpande, A. Gupta, J. F. Naughton, R. Ramakrishnan, and S. Sarawagi. On the computation of multidimensional aggregates. In *VLDB'96, Proceedings of 22th International Conference on Very Large Data Bases, September 3-6, 1996, Mumbai (Bombay), India*, pages 506–521, 1996. URL http://www.vldb.org/conf/1996/P506.PDF.

L. A. Barroso, J. Clidaras, and U. Hölzle. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines, Second Edition*. Synthesis Lectures on Computer Architecture. Morgan & Claypool Publishers, 2013. ISBN 9781627050098. doi: 10.2200/S00516ED2V01Y201306CAC024. URL http://dx.doi.org/10.2200/S00516ED2V01Y201306CAC024.

F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber. Bigtable: A distributed storage system for structured data. *ACM Trans. Comput. Syst.*, 26(2), 2008. doi: 10.1145/1365815.1365816. URL http://doi.acm.org/10.1145/1365815.1365816.

S. Chaudhuri and U. Dayal. An overview of data warehousing and OLAP technology. *SIGMOD Record*, 26(1):65–74, 1997. doi: 10.1145/248603.248616. URL http://doi.acm.org/10.1145/248603.248616.

M. Chen, S. Mao, and Y. Liu. Big data: A survey. *MONET*, 19(2):171–209, 2014. doi: 10.1007/s11036-013-0489-0. URL http://dx.doi.org/10.1007/s11036-013-0489-0.

E. F. Codd. A relational model of data for large shared data banks. *Commun. ACM*, 13(6):377–387, 1970. doi: 10.1145/362384.362685. URL http://doi.acm.org/10.1145/362384.362685.

E. F. Codd, S. B. Codd, and C. T. Salley. Providing olap to user-analysts: An it mandate. *Ann ArborMichigan*, 26:24, 1993.

J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. In *6th Symposium on Operating System Design and Implementation (OSDI 2004), San Francisco, California, USA, December 6-8, 2004*, pages 137–150, 2004. URL `http://www.usenix.org/events/osdi04/tech/dean.html`.

J. Desharnais, A. Grinenko, and B. Möller. Relational style laws and constructs of linear algebra. *Journal of Logical and Algebraic Methods in Programming*, 83(2):154–168, 2014. ISSN 2352-2208. doi: http://dx.doi.org/10.1016/j.jlap.2014.02.005.

A. Floratou, J. M. Patel, E. J. Shekita, and S. Tata. Column-oriented storage techniques for mapreduce. *PVLDB*, 4(7):419–429, 2011. URL `http://www.vldb.org/pvldb/vol4/p419-floratou.pdf`.

J. Ginsberg, M. H. Mohebbi, R. S. Patel, L. Brammer, M. S. Smolinski, and L. Brilliant. Detecting influenza epidemics using search engine query data. *Nature*, 457(7232):1012–1014, 2009.

G. Graefe. Query evaluation techniques for large databases. *ACM Comput. Surv.*, 25(2):73–170, 1993. doi: 10.1145/152610.152611. URL `http://doi.acm.org/10.1145/152610.152611`.

J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M. Venkatrao, F. Pellow, and H. Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub totals. *Data Min. Knowl. Discov.*, 1(1):29–53, 1997. doi: 10.1023/A:1009726021843. URL `http://dx.doi.org/10.1023/A:1009726021843`.

P. Groves and D. Knott. The “ big data ” revolution in healthcare. *M*, (January), 2013.

K. M. A. Hasan, T. Tsuji, and K. Higuchi. An efficient implementation for MOLAP basic data structure and its evaluation. In *Advances in Databases: Concepts, Systems and Applications, 12th International Conference on Database Systems for Advanced Applications, DASFAA 2007, Bangkok, Thailand, April 9-12, 2007, Proceedings*, pages 288–299, 2007. doi: 10.1007/978-3-540-71703-4_26. URL `http://dx.doi.org/10.1007/978-3-540-71703-4_26`.

D.E. Knuth. *The Art of Computer Programming : (1) Fundamental Algorithms, (2) Seminumerical Algorithms, (3) Sorting and Searching*. Addison/Wesley, 2nd edition, 1997/98. 3 volumes. First edition’s dates are: 1968 (volume 1), 1969 (volume 2) and 1973 (volume 3).

Hans-Joachim Lenz and Bernhard Thalheim. A formal framework of aggregation for the OLAP-OLTP model. *J. UCS*, 15(1):273–303, 2009. doi: 10.3217/jucs-015-01-0273. URL `http://dx.doi.org/10.3217/jucs-015-01-0273`.

X. Li, J. Han, and H. Gonzalez. High-dimensional OLAP: A minimal cubing approach. In *(e)Proceedings of the Thirtieth International Conference on Very Large Data Bases, Toronto, Canada, August 31 - September 3 2004*, pages 528–539, 2004. URL http://www.vldb.org/conf/2004/RS14P1.PDF.

J. Lin and C. Dyer. *Data-Intensive Text Processing with MapReduce*. Synthesis Lectures on Human Language Technologies. Morgan & Claypool Publishers, 2010. doi: 10.2200/S00274ED1V01Y201006HLT007. URL http://dx.doi.org/10.2200/S00274ED1V01Y201006HLT007.

H.D. Macedo and J. N. Oliveira. A linear algebra approach to OLAP. *Formal Aspects of Computing*, pages 1–25, 2014. ISSN 0934-5043. doi: 10.1007/s00165-014-0316-9. URL http://dx.doi.org/10.1007/s00165-014-0316-9.

Elizabeth Million. The hadamard product. 2007. URL http://buzzard.ups.edu/courses/2007spring/projects/million-paper.pdf.

P. Mishra and M. H. Eich. Join processing in relational databases. *ACM Comput. Surv.*, 24(1):63–113, 1992. doi: 10.1145/128762.128764. URL http://doi.acm.org/10.1145/128762.128764.

K. Morfonios, S. Konakas, Y. E. Ioannidis, and N. Kotsis. ROLAP implementations of the data cube. *ACM Comput. Surv.*, 39(4), 2007. doi: 10.1145/1287620.1287623. URL http://doi.acm.org/10.1145/1287620.1287623.

R. O. Nambiar and M. Poess. The making of TPC-DS. In *Proceedings of the 32nd International Conference on Very Large Data Bases, Seoul, Korea, September 12-15, 2006*, pages 1049–1058, 2006. URL http://www.vldb.org/conf/2006/p1049-othayoth.pdf.

R. O. Nambiar, M. Lanken, N. Wakou, F. Carman, and M. Majdalany. Transaction processing performance council (TPC): twenty years later - A look back, a look ahead. In *Performance Evaluation and Benchmarking, First TPC Technology Conference, TPCTC 2009, Lyon, France, August 24-28, 2009, Revised Selected Papers*, pages 1–10, 2009. doi: 10.1007/978-3-642-10424-4_1. URL http://dx.doi.org/10.1007/978-3-642-10424-4_1.

J.N. Oliveira. Transforming Data by Calculation. In *GTTSE'07*, volume 5235 of *LNCS*, pages 134–195. Springer, 2008. doi: 10.1007/978-3-540-88643-3_4.

J.N. Oliveira. Towards a linear algebra of programming. *Formal Aspects of Computing*, 24(4-6):433–458, 2012. URL http://dx.doi.org/10.1007/s00165-012-0240-9.

Alain Pirotte. A precise definition of basic relational notions and of the relational algebra. *SIGMOD Record*, 13(1):30–45, 1982. doi: 10.1145/984514.984516. URL http://doi.acm.org/10.1145/984514.984516.

M. Pöss and C. Floyd. New TPC benchmarks for decision support and web commerce. *SIGMOD Record*, 29(4):64–71, 2000. doi: 10.1145/369275.369291. URL http://doi.acm.org/10.1145/369275.369291.

P. Russom and IBM. Big data analytics. *TDWI Best Practices Report, Fourth Quarter*, (August): 38, 2011. URL http://www.mendeley.com/catalog/big-data-analytics-45/nftp://129.35.224.12/software/tw/Defining_Big_Data_through_3V_v.pdf.

S. Sarawagi and M. Stonebraker. Efficient organization of large multidimensional arrays. In *In Proceedings of the Tenth International Conference on DAta Engineering*, pages 328–336, 1994.

M. Silva. Sparse matrix storage revisited. In *Proceedings of the Second Conference on Computing Frontiers, 2005, Ischia, Italy, May 4-6, 2005*, pages 230–235, 2005. doi: 10.1145/1062261.1062299. URL http://doi.acm.org/10.1145/1062261.1062299.

M. Stonebraker, D. J. Abadi, A. Batkin, X. Chen, M. Cherniack, M. Ferreira, E. Lau, A. Lin, S. Madden, E. J. O'Neil, P. E. O'Neil, A. Rasin, N. Tran, and S. B. Zdonik. C-store: A column-oriented DBMS. In *Proceedings of the 31st International Conference on Very Large Data Bases, Trondheim, Norway, August 30 - September 2, 2005*, pages 553–564, 2005. URL http://www.vldb2005.org/program/paper/thu/p553-stonebraker.pdf.

A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, N. Zhang, S. Anthony, H. Liu, and R. Murthy. Hive - a petabyte scale data warehouse using hadoop. In *Proceedings of the 26th International Conference on Data Engineering, ICDE 2010, March 1-6, 2010, Long Beach, California, USA*, pages 996–1005, 2010. doi: 10.1109/ICDE.2010.5447738. URL http://dx.doi.org/10.1109/ICDE.2010.5447738.

TPC. *TPC Benchmark^{TM}C: Standard Specification, Revision 5.11*, 2009. URL http://www.tpc.org/tpc_documents_current_versions/pdf/tpc-c_v5-11.pdf.

TPC. *TPC Benchmark^{TM}E: Standard Specification, Version 1.13.0*, 2014a. URL http://www.tpc.org/tpc_documents_current_versions/pdf/tpce-v1.13.0.pdf.

TPC. *TPC Benchmark^{TM}H: Standard Specification, Revision 2.17.0*, 2014b. URL http://www.tpc.org/tpc_documents_current_versions/pdf/tpch2.17.1.pdf.

H. J. Watson and B. Wixom. The current state of business intelligence. *IEEE Computer*, 40(9):96–99, 2007. doi: 10.1109/MC.2007.331. URL http://doi.ieeecomputersociety.org/10.1109/MC.2007.331.

H. J. Watson, T. Ariyachandra, and R. J. Matyska Jr. Data warehousing stages of growth. *IS Management*, 18(3):42–50, 2001. doi: 10.1201/1078/43196.18.3.20010601/31289.6. URL http://dx.doi.org/10.1201/1078/43196.18.3.20010601/31289.6.

K. Wu, E. J. Otoo, and A. Shoshani. Optimizing bitmap indices with efficient compression. *ACM Trans. Database Syst.*, 31:1–38, March 2006. ISSN 0362-5915. doi: http://doi.acm.org/10.1145/1132863.1132864. URL http://doi.acm.org/10.1145/1132863.1132864.

S. Yevtushenko. *Computing and visualizing concept lattices*. PhD thesis, Darmstadt University of Technology, 2004. URL http://elib.tu-darmstadt.de/diss/000488.

Y. Zhao, P. Deshpande, and J. F. Naughton. An array-based algorithm for simultaneous multidimensional aggregates. In *SIGMOD 1997, Proceedings ACM SIGMOD International Conference on Management of Data, May 13-15, 1997, Tucson, Arizona, USA.*, pages 159–170, 1997. doi: 10.1145/253260.253288. URL http://doi.acm.org/10.1145/253260.253288.

A

LISTINGS

Listing 7 Generating the projection function,

```
    public ArrayList<Integer> generatePF(ArrayList<String> attributes){
        TreeMap<String, Integer> map = new TreeMap<>();
        ArrayList<Integer> rows = new ArrayList<>();
        int line = 0;

        for(String attr: attributes){
            if(map.containsKey(attr)){
                rows.add(rowsMeaning.get(attr));
            }else{
                map.put(name, line);
                rows.add(line);
                line++;
            }
        }
            return rows;
    }
```

Listing 8 Khatri-Rao product in MCSC

```
public ArrayList<Integer> khp(ArrayList<Integer> A,
                              ArrayList<Integer> B,
                              int attributesA,
                              int attributesB){
    ArrayList<Integer> res =
      new ArrayList<Integer>(attributesA*attributesB);
    K = B.length;
    for(int i = 0; i < K; i++){
        M = A.get(i);
        N = B.get(i);
        result.add(K*M + N);
    }
    return res;
}
```

Listing 9 Final Multiplication

```
public ArrayList<Integer> finalResult(Int projLinesA,
                                       Int projLinesB,
                                       ArrayList<Int> khr,
                                       ArrayList<Int> measure,
                                       ArrayList<Int> bang){

    int khrLines = projLinesA*projLinesB;
    ArrayList<Int> result  = new ArrayList<Int>(khrLines);

    for(int i = 0; i < khrLines; i++)
    {
        khrLines.add(i,0);
    }

    for(int i = 0; i< measure.length; i++)
    {
        int B = bang.get(i);
        int M = measure.get(i);
        int K = khr.get(i);
        if(B == 1){
            int old = result.get(K);
            result.add(k,old+M);
            }
    }
}
```

Listing 10 Query 1

```
select
   l_returnflag,
   l_linestatus,
   sum(l_quantity) as sum_qty,
   sum(l_extendedprice) as sum_base_price,
   sum(l_extendedprice * (1 - l_discount)) as sum_disc_price,
   sum(l_extendedprice * (1 - l_discount) * (1 + l_tax)) as sum_charge,
   avg(l_quantity) as avg_qty,
   avg(l_extendedprice) as avg_price,
   avg(l_discount) as avg_disc,
   count(*) as count_order
from
   lineitem
where
   l_shipdate <= date ``1997-12-01'' - interval ``95'' day
group by
   l_returnflag,
   l_linestatus
order by
   l_returnflag,
   l_linestatus
```

Listing 11 Query 3

```
select
    l_orderkey,
    sum(l_extendedprice * (1 - l_discount)) as revenue,
    o_orderdate,
    o_shippriority
from
    customer,
    orders,
    lineitem
where
    c_mktsegment = "MACHINERY"
    and c_custkey = o_custkey
    and l_orderkey = o_orderkey
    and o_orderdate < date "1995-03-10"
    and l_shipdate > date "1995-03-10"
group by
    l_orderkey,
    o_orderdate,
    o_shippriority
order by
    revenue desc,
    o_orderdate;
```

This dissertation was developed in the scope of the project LeanBigData(AE2014 - 0092).