Mestrado em Engenharia Informática e Sistemas

Experimental Evaluation of Big Data Querying Tools

Mário Miguel Lucas Rodrigues

Coimbra, dezembro, 2017

Mestrado em Engenharia Informática e Sistemas

Experimental Evaluation of Big Data Querying Tools

Dissertação apresentada para a obtenção do grau de Mestre em Informática e Sistemas

Especialização em Tecnologias da Informação e do Conhecimento

Autor

Mário Miguel Lucas Rodrigues

Orientadores

Prof. Doutor Jorge Bernardino

Departamento de Engenharia Informática e de Sistemas
Instituto Superior de Engenharia de Coimbra

Prof. Doutora Maribel Yasmina Santos

ALGORITMI Research Centre
Universidade do Minho

Coimbra, dezembro, 2017

Acknowledgements

This research is the final result of my thesis project to obtain the Master of Science degree in computer engineering with a specialization in Business Information and Knowledge Technology at the Institute of Engineering of Coimbra (ISEC), Polytechnic of Coimbra.

I would like to thank several people who supported me during this Master Thesis project. Firstly, I would like to leave a special thanks my supervisors Prof. Jorge Bernardino from the Polytechnic of Coimbra and Prof. Maribel Santos, from the University of Minho for all the support, time and constructive criticism during this process. This thesis would not have been possible without their big support, kindness, and patience.

I would like to thank the support of the staff from INCD (Infraestrutura Nacional de Computação Distribuída) that provided computation services and resources. Their cooperation was a great help in fulfilling my research goals.

Last but not least, I would also like to thank my parents and girlfriend for their support and patience while I did all of the work in the completion of this dissertation and my Master of Science in general, without them I wouldn't be able to do this.

"Success is not final, failure is not fatal: it is the courage to continue that counts."

Winston Churchill

Abstract

In the past years, Big Data has become a hot topic across several business areas. One of the main challenges regarding this concept is how to handle the massive volume and variety of data efficiently. Due to the notorious complexity of the data associated to the Big Data concept, usually motivated by data volume, efficient querying analysis mechanisms are mandatory for data analysis purposes. Motivated by the rapidly development of tools and frameworks for Big Data, there is much discussion about querying tools and, specifically, those more appropriated for specific analytical needs. This thesis describes and compares the main characteristics and architectures of the following popular Big Data analytical tools: Drill, Hive, HAWQ, Impala, Presto, and Spark. To test the performance of these Big Data analytical tools, we also describe the process to prepare, configure and manage an Hadoop Cluster to deploy these tools, having an environment able to evaluate their performance and identify in what scenarios they are suited. To perform this evaluation, we used TPC-H and TPC-DS benchmarks, where results have shown that in-memory processing tools like HAWQ, Impala and Presto provide better results and performance with low and medium datasets. However, the tools that presented slowest query execution times, especially Hive, seems to catch up with the best performing tools when we scale benchmark datasets.

Keywords: Big Data, Hadoop, SQL-on-Hadoop, Query Processing, Big Data Analytics.

Resumo

Nos últimos anos, o termo Big Data tornou-se um tópico bastanta debatido em várias áreas de negócio. Um dos principais desafios relacionados com este conceito é como lidar com o enorme volume e variedade de dados de forma eficiente. Devido à notória complexidade e volume de dados associados ao conceito de Big Data, são necessários mecanismos de consulta eficientes para fins de análise de dados. Motivado pelo rápido desenvolvimento de ferramentas e frameworks para Big Data, há muita discussão sobre ferramentas de consulta e, mais especificamente, quais são as mais apropriadas para necessidades analíticas específica. Esta dissertação descreve e compara as principais características e arquiteturas das seguintes conhecidas ferramentas analíticas para Big Data: Drill, HAWQ, Hive, Impala, Presto e Spark. Para testar o desempenho dessas ferramentas analíticas para Big Data, descrevemos também o processo de preparação, configuração e administração de um Cluster Hadoop para que possamos instalar e utilizar essas ferramentas, tendo um ambiente capaz de avaliar seu desempenho e identificar quais cenários mais adequados à sua utilização. Para realizar esta avaliação, utilizamos os benchmarks TPC-H e TPC-DS, onde os resultados mostraram que as ferramentas de processamento em memória como HAWQ, Impala e Presto apresentam melhores resultados e desempenho em datasets de dimensão baixa e média. No entanto, as ferramentas que apresentaram tempos de execuções mais lentas, especialmente o Hive, parecem apanhar as ferramentas de melhor desempenho quando aumentamos os datasets de referência.

Palavras-chave: Big Data, Hadoop, SQL-on-Hadoop, Query Processing, Big Data Analytics.

Table of Contents

List of Figures

List of Tables

Acronyms

BI – Business Intelligence

CDH – Cloudera Distributed Hadoop

CPU – Central Processing Unit

CSV – Comma-separated Values

ETL – Extract, Transform, Load

HDFS – Hadoop Distributed File System

HDP – Hortonworks Data Platform

HTTP– Hypertext Transfer Protocol

IoT – Internet of Things

IT – Information Technology

JSON – JavaScript Object Notation

MPP – Massive Parallel Processing

OLTP – Online Transaction Processing

RAM– Random Access Memory

SQL – Structured Query Language

SSH – Secure Shell

RDBMS – Relational Database Management Systems

TPC– Transaction Processing Performance Council

UI – User Interface

ORC – Optimized Row Columnar

XML – eXtensible Markup Language

1 Introduction

We live in a world of data since everything we do in our lives is leaving a footprint or trace. In this context, organizations are starting to realize the importance of taking advantage of data to support the decision-making process.

Because of the tremendous increase of the Internet usage, like social media, daily online operations and transactions from multiple sources, the amount of generated data has grown exponentially. This fact led to the arise of the Big Data concept that describes enormous datasets or large volumes of structured, semi-structured or unstructured data on several formats provided by several data sources.

Despite the complexity and the many challenges associated to the use of the Big Data concept, we cannot ignore the potential lying in it. It can provide support for analytics and for the identification of hidden patterns, which are very effective in defining business strategies.

1.1 Problem Statement

Nowadays Big Data still faces a lack of consensus and rigor in its definition and standardization. Common users associate the concept to complexity and hugely learning efforts. With the avalanche of Big Data, Hadoop was born out of a need to process huge volumes of data. We can see Hadoop as a synonym for Big Data due its capabilities to store and handle huge amounts of (unstructured) data. The powerful and ever-growing Hadoop ecosystem is, and has been, a clear leader in operational and exploratory data computation and analytics.

Due to this fact, we can consider Hadoop an essential tool in several aspects of Big Data, including big data analysis, but this architecture is still only accessible by a group of people that has knowledge of it.

It's necessary to transform the Big Data Analysis in a more accessible and less abstract concept, accessible to a high number of users. By making data analysis easier and faster it will result in a wider audience which can now expect to increase their returns on investment in big data. Structured Query Language (SQL) processing has gained significant attention, as many enterprise data management tools rely on SQL, and, also, many users are familiar and comfortable with it [1]. As a result, the number of SQL-on-Hadoop systems have increased significantly, addressing the concept and user related issues mentioned above.

The diverse number of Hadoop environment technologies, with a high number of associated components, foment common user rejection. Therefore, it is important to focus on Big Data Analysis and querying tools that can be used without the need of having the whole perception and knowledge of the Hadoop ecosystem.

In this work, a detailed study of some Big Data querying tools is presented, describing and comparing their main characteristics, describing how we can deploy them and install in a

real case scenario. We also perform a series of performance experiments using the TPC Benchmarks TPC-H and TPC-DS, in order to evaluate Drill, HAWQ, Hive, Presto and Spark, to conclude in which scenarios, they are suitable or ideal and those where is not recommended to use them.

In adition, this work also carried out with the aim of combating the complexity associated with the Big Data and associated concepts, so that a user who has never had contact with this area understands all aspects addressed, how can use these powerful tools.

1.2 Methodological Approach

In order to find which querying tools are suited to deal with Big Data Analysis, the starting point of this work consisted in a deep research of technological and theoretical Big Data related concepts in order to have a basic understanding of its environment. This background allowed us to reach the core subject of this work, the study of existing Big Data querying tools.

With these concepts in mind we were able to study several tools available in order to select the ones that in our opinion standed out and in which our work will focus on. To complement our initial selection, we performed an experimental evaluation using the TPC-H and TPC-DS benchmarks (detailed on subChapter 5.3 and) to determine which perform best and under what circunstances.

1.3 Main Contributions

The main contributions of this thesis are:

- A detailed description of selected Big Data querying tools, describing their main characteristics and architectures;
- Highlight and define the essential features that Big Data querying tools must have to efficiently process queries;
- Compare if the selected tools possess the essential identified features and requirements to be considered a viable option;
- Experimental evaluation of Big Data analytical tools performances using TPC-H and TPC-DS benchmarks;
- Description of the deployment and configurations of a close to a real case scenario environment in order to install the selected tools, also detailing how we used them.

These are, in a general way, the main contributions that have been made with the development of this work. We also contributed to the existing literature:

- We published a paper intitled “Describing and Comparing Big Data Querying tools” (Appendix A) that was submitted and published on the 5th World Conference on Information Systems and Technologies - WorldCIST’17, and integrated as part of the book Recent Advances in Information Systems and Technologies, vol 569, pp. 115-

124, ISBN: 978-3-319-56534-7, Springer, Cham (DOI: 10.1007/978-3-319-56535-4_12).

- This work also received an invitation to submit an extended version to Data Mining and Knowledge Discovery, a triannual peer-reviewed scientific journal focusing on data mining, published by Springer Science and Business MediaTheoretical contributions. We accepted the invitation resulting in the submission of the paper intitled "Big Data Analytical Tools: An Experimental Performance Evaluation", extending the previously work, presenting a performance evaluation made through the TPC-H benchmark (Appendix B).
- Finally, we submitted a paper intitled "Experimental evaluation of Big Data Analytical Tools" to CAISE'18, 30th International Conference on Advanced Information Systems Engineering where we evaluate the previously referenced tools using ad hoc queries part of the TPC-H and TPC-DS (Appendix C).

The final decision of these last two papers is still pending.

1.4 Outline

This thesis is structured as follows. In Chapter 3, we analyze the related work previously performed, also presenting a conceptual and technological framework, describing the relevant Big Data related concepts and technologies. Chapter 4 will present the chosen set of Big Data querying tools, result of an extensive research, where we describe the tools and architectures, also defining some ideal requirements that, in our opinion a suited tool must have.

Through these defined requirements we also compare the presented tools to find if they are compliant with requirements. Chapter 5 describes the methodology of the performed practical approach, describing implementation environment, decisions made, and the benchmarks performed. In Chapter 6 we present the experimental results obtained and respective analysis.

Finally, in Chapter 7, we will summarize the main findings of this research, discuss the limitations of this research and point out some future work.

2 Big Data Conceptual and Technological Frameworks

In this chapter, we present concepts and technologies related to the subject of this work, presenting a conceptual and technological framework, where we describe the relevant Big Data related concepts and technologies, providing a base of knowledge, so that even users that never had contact with this area can understand.

2.1 Conceptual Framework

In this, we present the conceptual concepts that we need to understand Big Data, presenting the definition and its key characteristcs, advantages, challenges and other related concepts.

2.1.1 The Big Data Concept

Under the increase of global data generated, the term of Big Data is mainly used to describe enormous datasets or large volumes of structured, semi-structured or unstructured data. The main importance of Big Data consists in the potential to improve efficiency when using a large volume of data, with several formats.

The data can be collected by smartphones, social networks, web server logs, traffic flow sensors, satellite images, broadcast audio streams, banking transactions, MP3s music, Global Positioning System (GPS) trails to name only a few. If Big Data is defined properly and used accordingly, organizations can have a better view on their business and what can be improved in different areas like sales, manufacturing processes, client management and acquisition. Based on the works [2] and [3] we can consider some examples of effective use of Big Data:

- In information technology in order to improve security and troubleshooting by analysing the patterns in the existing logs;
- In customer service by using information from call centers, or other client records in order to get the customer profile and enhance customer satisfaction through service customization;
- In the improvement of services and products using social media content, this information can reveal costumer preferences to address a larger number of clients;
- In fraud and outlier's detection in transactions of any industry (often used in banking transactions);
- In risk assessment by analyzing information from the transactions on the financial market.

In 2001, Doug Laney, a Gartner Analyst came up with the famous three Vs model that clarifies the nature of Big Data (illustrated on Figure 1):

- **Volume:** Refers to the ability to process large amounts of information (it is common to reach Terabytes, Petabytes, Exabytes or Zettabytes), that can't be handle in conventional relational databases infrastructures;
- **Velocity:** Refers to the need of processing the data in a short period, organizations once considered that data with some hours or days were recent, but now their definition of recent have been moving towards the near real-time;
- **Variety:** Refers to the variety of data sources. Many data is unstructured and not always fits the common relational structures (rows and columns), since it may be documents, text, image data, or raw feed directly from a sensor.

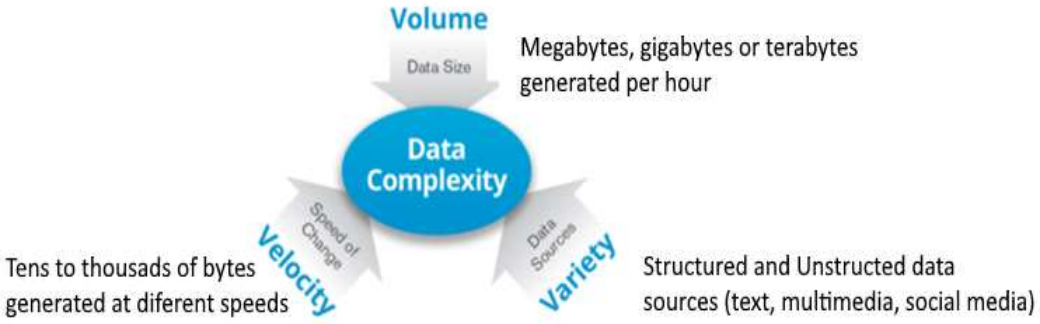


Figure 1. 3V's model. Adapted from [4] and [5].

With the growth of interest of the scientific and technological community on the Big Data concept in the past years, it was added two more characteristics, originating the 5V's model (see Figure 2):

- **Veracity:** Having a lot of data coming at high speed is worthless if the data isn't accurate, therefore interested users must ensure that the data is trustable as well as the analyses performed on the data are correct. Organizations need to develop improvements in the data management area (Data Governance) to ensure the veracity of data provided by new sources;
- **Value:** The data should present valuable results to aid in decision making. To extract value from there is a need to determine its veracity.

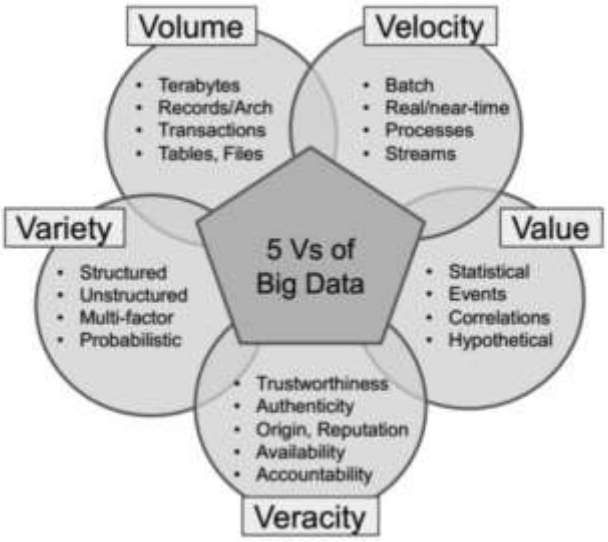


Figure 2. 5V's model. Retrieved from [4].

The original 3V model clearly differentiates Big Data from any other data, several analysts, enthusiasts and researchers created other models, like the 7V's and 10V's model.

These models complement the previous described models adding characteristics like visualization and variabiability, that refer to the visualization of data to extract value and inconsistencies in Big data. For more details in these models and it's characteristics, please refer to [6], [7] or [8]. All the characterstics define the nature of Big Data, but what really matters is value, other characteristics of big data are meaningless if we don't extract value from data.

2.1.1.1 Big Data Advantages and Challenges

The growing tendency of exploring and taking advantage of the Big Data revolution is creating new ways to gather and analyze information.

As the volume of data continues to grow, its potential for business seems to be growing exponentially as Big Data management solutions evolve allowing companies to turn raw data into relevant trends, predictions, and projections, bringing with it a new set of benefits [9], [10]:

- **Risk mitigation:** Mitigating risk by optimizing the complex decisions of unplanned events more rapidly;
- **Faster and better decision making:** Analytics over data has always involved to improve decision making;
- **Real time analysis:** Identifying the root causes of failures and issues in real time;
- **Customize user or customer experiences:** Big Data can help to unlock new business opportunities and identify areas for optimization;
- **Cost reduction:** Big data technologies like Hadoop and cloud-based analytics can provide lower costs compared to traditional architectures like Data Warehouses;

- **Pattern discovery and profiling:** The analysis and interpretation of users or customer behaviors offers the opportunity of acquiring new products/services and prediction of scenarios;

Though the benefits of Big Data are factual and substantial, due to its characteristics, some challenges and concerns arise, like issues in security, privacy and monitoring. These concerns must be addressed, to fully take advantage of this concept.

Based on the works of [4], [10] and [11] we've outlined some of what we've consider key challenges:

- **Heterogenity:** The heterogenity resulting from multiple data sources brings consequences on data integration and analysis, since the unstructured natured of data sources presents several challenges regarding transformations to support analytical tasks;
- **Privacy and security:** Privacy and security concerns individuals, since they have the right, according to International Communication Union, to control the information that may be disclosed. Information about individuals in the scope of Big Data makes individual privacy a delicate problem. For companies the privacy issue is more related to sensitive business information like financial data, clients list or projects;
- **Sharing data:** To get some value of data, it must be accessible to the general public. Nowadays every person has access to information about everything, but this is only possible if everyone agrees to share information. For example, regarding companies, most of them refuse to share their information due to competitiveness;
- **Identify the right information:** The quantity of data that is being stored is not always one hundred percent useful which makes it difficult for companies to identify the right data and determine the best way to use it;
- **Choosing architecture based on cost and performance:** Choosing architecture and building an appropriate big data solution represents a challenge. There is a need to rethink storage devices, architectures, mechanisms and networks, in order to achive efficient input/output, data accessibility and data tansmission;
- **Lack of Specialists and skill requirements:** To work with Big Data, workers must have a diverse skill set, that includes skills in research, analysis, interpretation and creativity. If companies or entities try to analyze the data with their own employees that have minimum skills or don't have them at all, this can lead to wrong conclusions. This can be bypassed with the implementation of software solutions which do not require special skills;
- **Acquiring data:** To find answers for questions that we want answered, there is a need to carefully select the data sources and define how we can discover value in the data provided.
- **Timeliness:** There are many situations in which the result of the analysis is required immediately. Since we deal with large datasets, consequently its processing will take longer to process and retrieve results.

2.1.1.2 Big Data Formats and Sources

In addition to the exponential data growth, data have also become increasingly sparse and semi-structured in nature. Data structures can be classified into the following main types:

- **Structured:** Structured data refers to information with a high degree of organization that has a defined length and format and can be easily stored in a relational database. Is seamless and readily searchable by simple, straightforward search engine algorithms or other search operations. Examples of structured data include numbers, dates, and groups of words and numbers;
- **Semi-Structured:** Semi-structured is a form of structured data that does not conform with the formal structure of data models associated with relational databases, but it has some organizational properties that make it easier to analyze. Usually this kind of data contains tags or other markers to separate semantic elements and enforce hierarchies of records and fields within the data. With some process these types of data can be stored in a relational database. Some semi-structured examples are commonly known by user like CSV, XML and JSON;
- **Unstructured**: Unstructured data refers to information that doesn't have a pre-defined data model and/or is not organized in a predefined manner. IBM estimates 80% of the world's data is unstructured. Examples of unstructured data include documents, images, videos, mobile data (communications, location, text messages, among others), data generated from social media platforms like YouTube, Facebook, etc.

2.1.1.3 Big Data and IoT

IoT is about devices, data and connectivity. The real value of Internet of Things is about creating smarter products, delivering intelligent insights and providing new business outcomes through real-time accurate data sensing and wireless transmission of that data to Web applications and servers connected to the Internet.

The Internet of Things generally refers to many different "things" in everyday objects connected to the internet, allowing communication between them, enabling these to send and receive data. IoT devices can be sensors, databases, other devices or software. Sensors could include pacemakers, location identifiers, such as global positioning system (GPS), and individual identification devices, such as radio-frequency identification (RFID) tags. The more objects that are connected, the more powerful the IoT becomes. Current trends in technology, such as increased adoption of wearable computers and other IoT devices, are allowing for unprecedented access to massive amounts of heterogeneous data, here's where IoT intersects with Big Data. The volume of data attributable to the Internet of Things is substantial. As sensors interact with the world, these "things" generate volumes and volumes of data.

As a result, digital processing becomes a requirement of feasibility. The velocity of data associated with the Internet of Things, compared with traditional transaction processing, explodes as sensors can continuously capture data. Also, the variety grows as the types of

sensors and the different sources of data expand. In sum, a large number of IoT devices with their wide range of sensors generate a huge volume of data which has variety, volume and velocity to qualify as Big Data.

Without the proper data gathering, it would be impossible to sort through all the information flowing in from embedded sensors, meaning that without Big Data, the Internet of Things can offer an enterprise little more than noise. In our opinion, we think data is the fuel that nurture IoT, consequently Big data and IoT basically go hand in hand. It's not just that Big Data and the IoT help each other, they also greatly impact each other.

The more the Internet of Things grows, the more demands are placed in terms of Big Data capabilities. Big Data technologies need to be augmented to effectively store, manage and extract value from continuous streams of sensor data, unleashing the potential of real-time Big Data analytics (see 2.1.2.1).

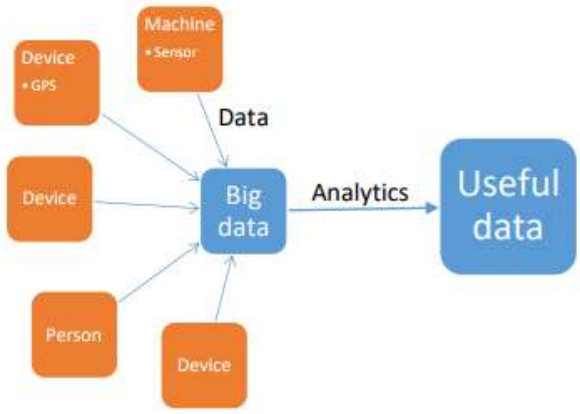


Figure 3. IoT and Big Data relationship. Retrieved from [12].

2.1.2 Big Data, Business Intelligence, and Decision Making

In the literature, Big Data, Business Intelligence, and Decision Making are considered as three strongly related research areas. Business Intelligence (BI) has received widespread attention over the past two decades in both the academic and the business communities. BI is a technology-driven process for analyzing data and presenting actionable information to help executives, managers and other corporate end users make informed business decisions.

BI is action, it means engaging with information, whether regular-size or Big Data, and making something meaningful happen through it, providing analysis capabilities of raw data to gain valuable business insights. Consequently, we believe integrating Big Data Analytics with BI systems is an important step toward gaining full return on investment.

These two concepts can be considered highly complementary, advanced analytics on Big Data can provide the deeper, exploratory perspective on the data, while BI systems provide a more structured user experience, through dashboard visualization, reporting, performance management metrics, among others, making advanced analytics actionable (see Figure 4).

However, traditional static Business Intelligence tools can no longer be efficient in the case of Big Data applications, facing new challenges because of dramatic development of Big

Data. Due to these facts, how we use big data analytics to enhance BI becomes a relevant question, in order to make the best options in decision making processes.

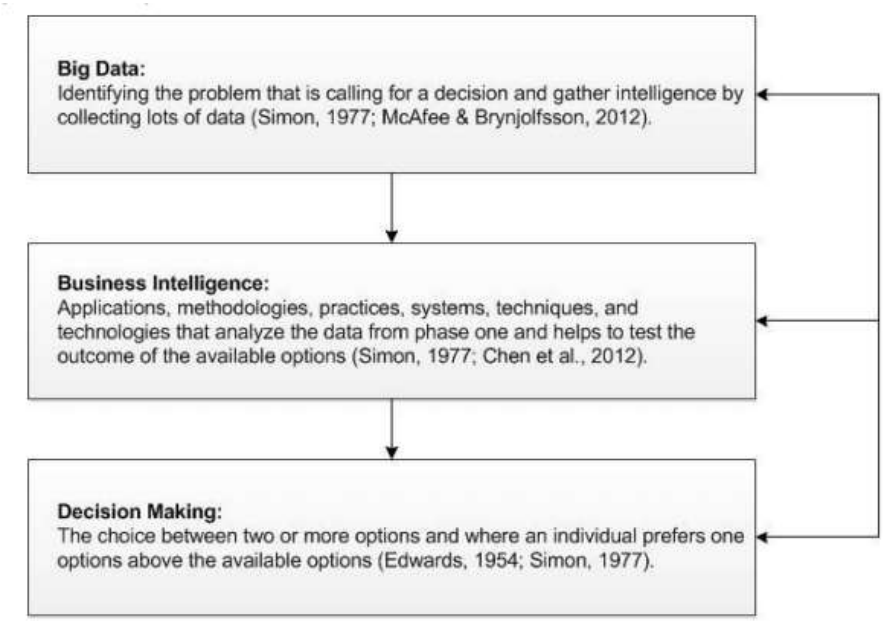


Figure 4. Big Data, Business Intelligence and Decision Making. Retrieved from [9].

2.1.2.1 Big Data Analytics

Analytics is not a new concept, there are many analytic technologies that have been available and applied for many years, even before the rise of Big Data concept, like regression analysis, simulations and machine learning.

Similar to these technologies, Big Data analytics applies techniques, tools, technologies and processes for making sense and extract knowledge out of Big Data. Big Data is worthless if we can not extract value from it. Big Data analytics can be defined as process of examining large and diverse datasets, include different types such as structured/unstructured and streaming/batch, and different sizes from terabytes to zettabytes.

Analytics applied on Big Data can be used to discover hidden patterns, unknown correlations, market trends, customer preferences and other useful information. With this information, analysts, researchers, and business users are able to make better and faster decisions, disvovering new revenue opportunities and maket advantages, using data that was previously inaccessible or unusable.

To perform this kind of analysis, there are a number of practices and technologies that can be applied, including data mining, predictive analytics, natural language processing, and artificial intelligence such as machine learning, decision trees, and neural networks. The Hadoop data framework (detailed in subChapter 2.2.2) has been a great tool for Big Data analysis, but it is still only accessible by a limited group of people, because of the huge efforts needed to learn its unique architecture.

To surpass this limitation, we believe that SQL-on-Hadoop is an important concept in the context of making Big Data analysis accessible to more people and making data analysis

easier and faster without the requirement of knowing the internals of Hadoop and Big Data concepts.

In a nutshell, Big Data Analytics can be defined as the use of advanced analytic techniques on Big Data sets, refers to the process of collecting, organizing and analyzing large sets of data. We briefly review some Big Data analytical techniques for structured and unstructured data, based on [3], the following techniques represent a relevant subset of the tools available for big data analytics:

- **Text analytics:** Text analytics or text mining refers to techniques that extract information from textual data, enabling the convertion of large volumes of human generated text into meaningful summaries. Social network feeds, emails, blogs, online forums, survey responses, corporate documents, news, and call center logs are examples where this technique can be applied;
- **Audio analytics:** Audio analytics (or speech analytics, if applied on human spoken language) analyze and extract information from unstructured audio data. Currently, customer call centers and healthcare are the primary application areas of audio analytics [3]. Call centers use audio analytics for efficient analysis of recorded calls. In healthcare, audio analytics support diagnosis and treatment of certain medical conditions that affect the patient's communication patterns These techniques can help improve customer experience, gain insight into customer behavior and identify product or service issues, among many other tasks;
- **Video analytics:** Video analytics involves a variety of techniques to monitor, analyze, and extract meaningful information from video streams. A key challenge is the sheer size of video data. To put this into perspective, one second of a high-definition video, in terms of size, is equivalent to over 2000 pages of text. Now consider that 100 hours of video are uploaded to YouTube every minute [3]. Big data technologies turn this challenge into opportunity, applying video analytics to draw intelligence from thousands of hours of video;
- **Social media analytics:** Motivated by the adoption of social media by consumers worldwide, Social media analytics refer to the analysis of structured and unstructured data from social media channels. Social media is a broad term encompassing a variety of online platforms, like social networks (e.g., Facebook and LinkedIn), blogs and media sharing platforms (e.g., Instagram and YouTube). The research on social media analytics is used across several areas, including psychology, sociology, computer science, mathematics, physics, economics and mainly in marketing;
- **Predictive analytics:** Predictive analytics comprise a variety of techniques that predict future outcomes based on historical and current data. At its core, it seeks to uncover patterns and capture relationships in data. It can be applied over almost all disciplines, from failures prediction to customer and sales prediction, through several techniques based on statistical methods.

2.1.2.2 Big Data Life Cycle

Big Data analytics can be viewed as a subprocess in the overall process of knowledge extraction from Big Data. The overall pratical process of extracting knowledge from big data can be broken down into five stages.

Before these five stages, there are two other stages that focus on what data can be used to generate knowledge. These two initial stages will influence deeply the whole Big Data Analytics life cycle:

- **Business case evaluation:** An evaluation of a Big Data analytics business case helps decision-makers understand the business resources that will need to be utilized and which business challenges the analysis will tackle. This evaluation should generate business requirements that will determine whether the business problems are being addressed or solved through Big Data insights;
- **Data Identification:** The Data Identification stage shown in is dedicated to identifying the datasets required for analysis and their sources. Evaluated data should be useful to discover value in the data provided. Identifying a wider variety of data sources may increase the probability of finding hidden patterns and correlations.

In a very summarized way, when we deal with Big Data, data is retrieved from the data sources and then loaded into a data platform or storage system. Then metadata is applied to create a structure to store that data. After the data is structured, its transformed and analyzed. All these steps are performed over the following five stages [3]:

- **Data Acquisition and Recording:** Data is gathered from all the data sources that were identified during the previous data identification stage;
- **Extraction, Cleaning and Annotation:** The collected data is mostly not in the format required for processing, consequently we cannot effectively analyse data in this manner. This stage pulls out the essential information from the primary sources and presents it in a structured form;
- **Data Integration, Aggregation and Representation:** Given the heterogenity of Big Data, it is not enough to capture it and save in our repository. We need to integrate, aggregate and represent data effectively.
- **Modeling and Analysis:** This stage involves the data analysts who can apply Query processing, data nining and analysis techniques over data in order to extract knowledge;
- **Interpretation:** The ability to analyze massive amounts of data and find useful insights carries little value if the only ones that can interpret the results are the analysts. Business users need to be able to understand the results in order to obtain value from the analysis.

The described five stages of the life cycle form two main sub-processes, Data Management and Analytics (see Figure 5).

Data management involves processes and supporting technologies to store data, prepare and retrieve it for analysis. Analytics, on the other hand, refers to techniques used to analyze and acquire intelligence from Big Data.

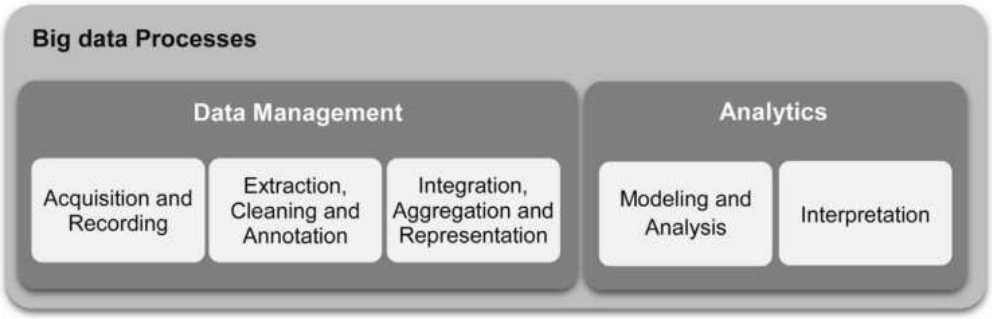


Figure 5. Processes for extracting insights from Big Data. Retrieved from [3].

2.1.3 Traditional and Big Data

The amount of data being generated daily is constantly increasing, forming huge datasets, pushing the limits of traditional data processing technologies. This continuous growing of data and the need to organize and search it has been revealing some lack of performance in the traditional RDBMS, not just because of the volume of Big Data, but also its characteristics.

Most of the data comes in a semi-structured or unstructured format from several sources like social media, audio, video, texts, and emails.

Traditional relational databases can't categorize unstructured data, since they are designed and structured to accommodate structured data. Although in Big Data environments, data is first collected and loaded to a certain storage system, a metadata layer is applied and then a structure is created. There is no need to start by transforming data to properly fit a relational model, as transformations only occur after having everything stored in efficient storage systems.

This represents a variation from the traditional Extraction, Transformation and Loading (ETL) approach to an Extraction, Loading and Transformation (ELT) [10]. RDBMS are massively used in the world, but the schema-rigidity (forcing correct design of the model at the starting point of a project) and their performance for big volumes of data, are becoming a bottleneck. Also, Big Data is generated at a very high velocity, consequently RDBMS lacks in high velocity, since it's designed for steady data retention rather than rapid growth.

To efficiently handle Big Data, novel ways for performing efficient analyses using the available computing resources must be employed, leading to the rise to new distributed Big Data engines. Also, methods for querying and mining Big Data are fundamentally different from traditional statistical analysis on small samples, since Big Data is often noisy, dynamic, heterogeneous, and untrustworthy.

2.1.3.1 Data Warehouses and Big Data

In the scope of Big Data and BI, frequently it's the differences between the concept of traditional Data Warehouses and Big Data that is not clear for more inexperienced users.

It is worth pointing out, Big Data is not a replacement for exisiting relational technologies like Data Warehouses. A Data Warehouse can be defined as a large data storage system, developed to store large amounts of data for analytical processing, helping in decision making and application of data mining algorithms.

It can store history of a business, collecting data from several sources, consequently it can easily grow to terabyte or even petabytes of size. On the other hand, the concept of Big Data presumes the storing of astronomical quantities of different data types which may be unstructured.

A Data Warehouse (DW) stores cleaned content, high quality and filtered data who suffered processing and filtering processes, thanks to ETL processes and other functions aiming to ensure that the information is relevant.

2.1.3.2 Traditional BI and Big Data Analytics

Although Big Data and Business Intelligence are two technologies used to analyze data to help companies in the decision-making process, there are differences them. They differ in the way they work as much as in the type of data they analyze.

Traditional BI works on structured data, usually stored at a Data Wharehouse (since it can be stored stored in columns and rows), usually providing answers to previsously defined questions, being unable to provide new insights or create the unanticipated findings decision makers are searching for.

On the other hand, Big Data analytics use raw data incomprehensively varied, (data with no pre-defined data model) to apply analysis tools and techniques to help present the data in meaningful ways.

When we dive into Big Data and Big Data Analytics, we can perform analysis at a much deeper level. This doesn't mean that previsously defined questions can not be applied in Big Data, as we seen previously the traditional BI concept can be considered highly complementary.

Also, the main difference towards traditional BI is the discovery of what questions can be asked, that we didn't know that had value, leading to discover new revenue opportunities, market advantages and hidden patterns.

Figure 6 presents examples of use to the two approaches and it complements this comparison, where it is visible that Big Data Analysis explores what questions could be ask after the retrieval of several types of raw data, from different sources, providing new insights and questions that can lead to deeper knowledge and unanticipated findings. On the traditional approach, the questions were previously defined with foundation in historical data and past results.

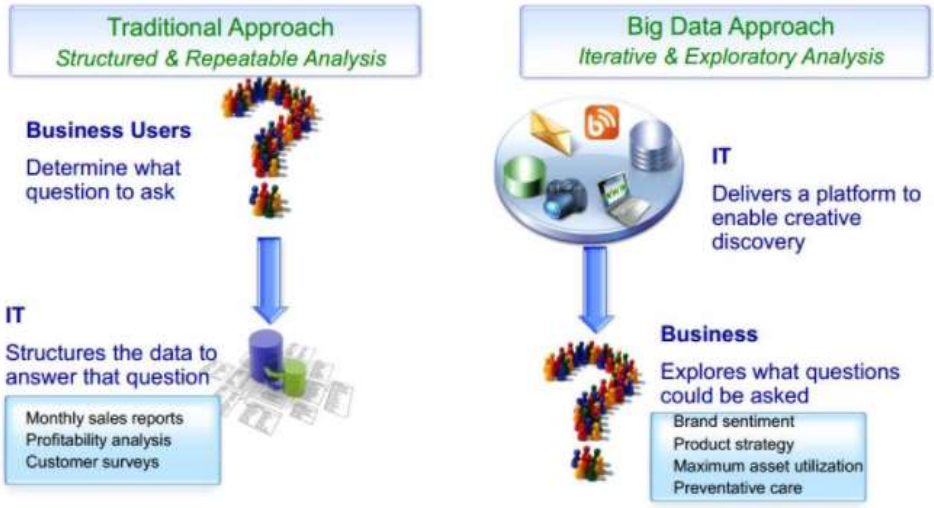


Figure 6. Traditional BI VS Big Data analysis. Retrieved from [11].

2.2 Technological Framework

To complement the previous conceptual framework, we now present the key technological concepts that support Big Data, among other concepts we highlight the detailed description of the Hadoop Ecosystem, its key components and also the category of SQL-on-Hadoop tools.

2.2.1 Computing Clusters

Clusters are essential when we work with Big Data, since its required large-scale processing. The term Computing Cluster generally refers to a set of interconnected computers that execute distributed, compute-heavy applications [13]. In many aspects, they can be viewed as a single system, varying in size, with CPU counts ranging from tens to hundred-thousand and storage capacity up to Petabytes and more.

The basic unit of a cluster is a single computer, also called a node. In the context of this thesis, a (very small) computing cluster is used to conduct our experiences described in subChapter 4.1. Clusters can be broadly categorized into two major classes:

- **Heterogenous:** Computers that span a wide range of hardware configurations and architectures. Heterogenous clusters naturally occur if older and newer hardware are used together.

- **Homogenous:** All underlying computers have the same or very similar hardware and system architecture. Such clusters are therefore easier to maintain.

Independent of this distinction, a cluster can be built either upon specialized hardware to maximize energy efficiency and performance, or on so-called commodity hardware that is cheaper to buy and uses components not different than in systems that are sold to end-users.

In the Big Data scope in terms of clusters, they are referenced as Hadoop clusters, a special type of cluster, specifically designed for storing and analyzing huge amounts of data,

known for boosting the speed of data analysis applications using commodity hardware, being easily scalable.

An Hadoop cluster is essentially a computational cluster that distributes the data analysis workload across multiple cluster nodes that work to process the data in parallel, ideally suited to handle Big Data.

2.2.2 Apache Hadoop Framework

Hadoop is one of the technologies that is immediately highlighted when discussing Big Data. Apache Hadoop has roots in the open source community. Developed by Apache Foundation, is one of the most widely heralded new platforms for managing big data.

It has been inspired on Google File System (GFS) and it was designed to avoid the low performance and the complexity encountered when processing and analyzing Big Data using traditional technologies, providing an environment based on distributed storage and computation across clusters of computers. The concept of batch processing is relevant to understand Hadoop's objectives, since these do not go through reduced response times, real-time access or high performance in transaction processing, but through the discovery and analysis of results in datasets which were once almost impossible to process, although several components have been integrated to offer low latency data access and stream processing [4].

Hadoop platform is based on two main subcomponents: the Hadoop Distributed File System (HDFS), and the MapReduce framework (explained in detail below). Using HDFS, data is stored in a replicated and distributed fashion through the several nodes of an Hadoop cluster. This replication of data across the cluster provides fault tolerance and resilience against server failure, making the system more robust and less prone to failures [14], [15].

We can also consider Hadoop as cost-effective since it was designed to run on commodity hardware. Commodity hardware is simply affordable hardware, which means that to build an Hadoop Cluster we don’t have to be tied up to expensive offerings from specialized vendors, we can use commonly available hardware (this does not mean that commodity is low-end hardware). Being commodity hardware more conducive to failures, the replication logic mentioned previously assures fault-tolerance and self-recover.

Beyond HDFS and Map Reduce, the Hadoop ecosystem comprises a set of components distributed through several layers. Figure 7 ilustrates Hadoop’s architecture, specifying core components (since it is significantly extensive to identify all components) by the several layers.

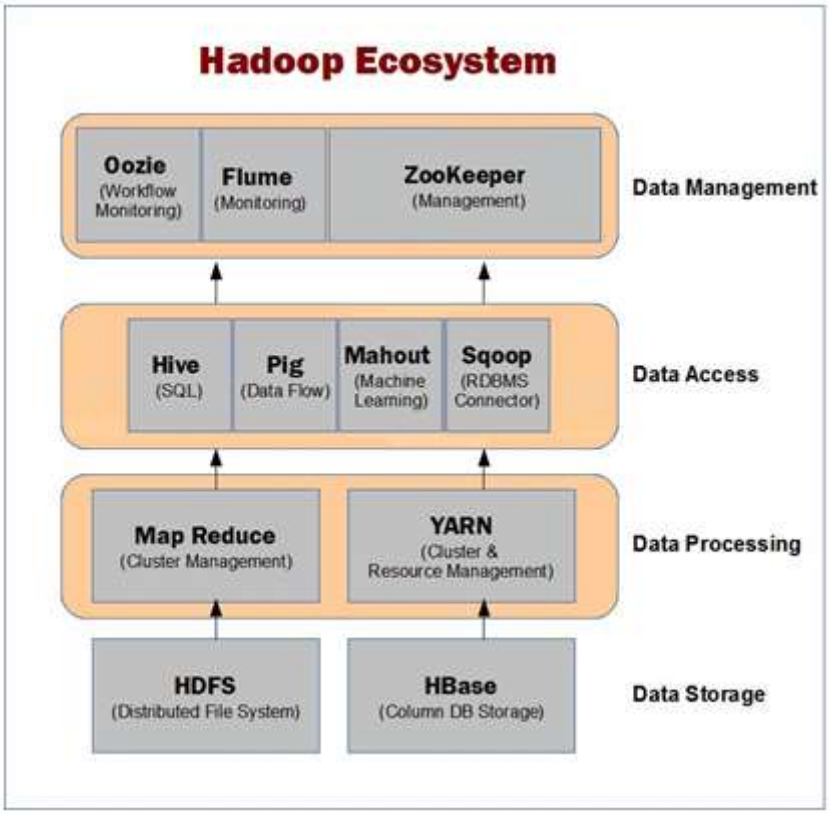


Figure 7. Hadoop ecosystem. Based on [14].

To complement the figure above, in Table 1 we describe the core components of Hadoop. Still related to Hadoop, there are several security projects for administration, authentication, authorization, audit and data protection.

Kerberos, Apache Knox and Apache Ranger are highlighted, to assure a secure Hadoop environment.

Table 1. Hadoop feature components.

Hadoop Layer	***Components***	***Description***
Data Storage	**HDFS**	Apache Hadoop Distributed File System.
	HBase	A non-relational distributed database intended for distributed scalable storage.
Data Processing	**MapReduce**	Framework that allows the parallel processing of vast datasets. MapReduce 2 is now developed on top of YARN, performing data processing via YARN.
	YARN	Responsible for the managing tasks and for the allocation of cluster resources.
	Tez	Also developed on top of YARN, Tez offers performance and flexibility of in batch and interactive processing. Its better performance led to its use in Hive, replacing the use of MapReduce.
Data Access	**Hive**	Data warehouse system for *ad hoc* queries and analysis of large datasets, also providing table and

		storage management services (described with more detail in subChapter 4.3)
	Pig	Scripting platform for analyzing large datasets
	Mahout	Machine learning and data mining library that allows the use of filtering, clustering and classification algorithms on datasets.
	Sqoop	Transfers data efficiently between Hadoop and relational databases.
Data Management	**Oozie**	Tool responsible for organizing and schedule tasks performed by users.
	Flume	Tool capable of retrieving data from several datasources, e.g Web Server logs.
	Zookeeper	Responsible for the coordination of distributed applications.

2.2.2.1 Hadoop Distributed File System (HDFS)

HDFS is the Hadoop data storage system. It supports up to hundreds of nodes in a cluster and provides a cost-effective and reliable storage capability [5], supporting the already mentioned data replication. When data is pushed into HDFS it automatically splits up into multiple blocks and stores/replicates the data thus ensuring high availability and fault tolerance [14], comprising two important components, NameNode and DataNodes.

These two components work in a Master-Slave architecture model, in which NameNode is Master and DataNode is Slave. HDFS NameNode as Master coordinates the file access and holds information (Metadata) that maps file names to block ids and to the DataNodes that hold replicas of them, while DataNodes are responsible for providing the actual storage, storing the blocks serving read and write requests on the data they hold.

Hadoop clusters usually comprises hundreds of nodes. If we install HDFS, one node will have the role of NameNode, a node as Secondary Namenode (in case Namenode fails). The remaining nodes can host DataNodes, the available HDFS storage will be the sum of all DataNodes disk capacity allocated to HDFS use.

2.2.2.2 MapReduce

It is important to first understand what makes MapReduce fast, since it was one of the first steps to efficiently manage Big Data, introducing cost effective and efficient mechanisms to process massive volumes of data stored in HDFS. Very briefly, MapReduce divides a task into small sub-tasks, which can be managed and distributed by multiple nodes, working together to obtain a certain result. It’s based on the divide and conquer method, dividing a complex problem into many simpler problems and then combining each simpler problem into an overall solution to the main problem [10].

The framework manages all the details of data-passing such as issuing tasks, verifying task completion, and copying data around the cluster between the nodes, being able to scale

the process over hundreds, thousands, or even tens of thousands of machines. To achieve this, MapReduce parading works with base two stages, map (takes the input data), and reduce (outputs the final result):

- The map or mappers job processes the input data (stored in HDFS in the form of file or directory), dividing the input data into blocks on the form of key-value pairs;
- After this split, the MapReduce framework sent all the key-value pairs into the Mapper that processes each of them individually, throughout several parallel map tasks across the cluster, generating as output intermediate key value pairs, that will be collected to perform sorts and grouping by key. The result will be many keys with a list of all associated values;
- Next, the reduce stage will process intermediate data, for each unique key. The Reducer job will aggregate the values associated to each key, producing as output a new set of key-value pairs which will be stored in the HDFS.

To better understand Figure 8 ilustrates the described process of the map and reduce stages.

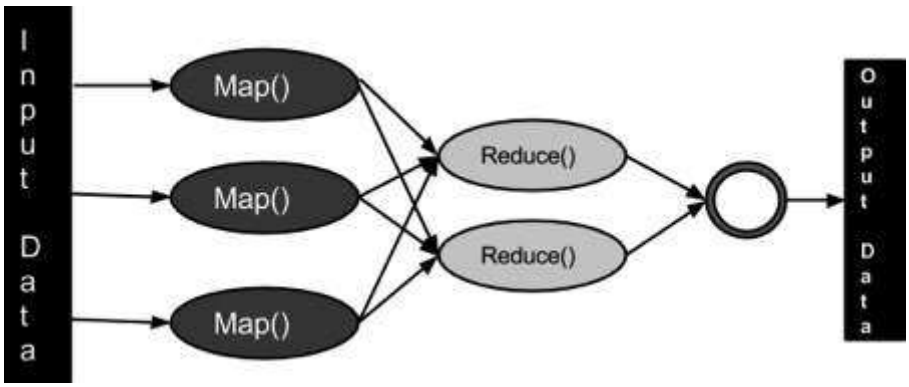


Figure 8. MapReduce process. Retrieved from [16].

The way MapReduce works is not immediate to understand, and to help visualize the concept beyond Figure 8, we consider a realistic example.

Supposing we manage an e-commerce website for a very large retailer, which has large stock of distinct products, and our website receives thousands of visitors every day. Over time, we have stored a vast collection of search terms that our visitors have typed in on our website.

The marketing department wants to know what customers are interested in, so we need to start discovering value from this mountain of information. This is a modest example, but we believe it is applied over several websites that we visit in real life. Imagining we simply want a sorted list of search terms applying MapReduce:

- The data should ideally be broken into numerous blocks;
- Each file will be distributed to a different node;

- On each node, the Map step will produce a list, consisting of each word in the file along with how many times it appears, e.g Skate: 4992120;
- The Reduce step will then consolidate all of the results from the Map step, producing a list of all search terms and the total number of times they appeared across all of the files.

In order to get better results, Hadoop tries to run the map phase on the same node as where the input data is stored on HDFS. This way, network bandwidth is not wasted. In the case of that node is busy running other map tasks, Hadoop chooses another node in the cluster with the replicated data. The name for that process is called data locality optimization [17].

In terms of nodes, in MapReduce there are also two types of nodes, JobTracker and TaskTrackers. JobTrackers, runs on the same node as HDFS Namenode, schedules jobs and distributes tasks across slaves or TaskTrackers.

To insure execution reliability, the JobTracker monitors the status of the slave nodes and re-assigns tasks when they fail. These components are part of the earlier version of MapReduce framework in Hadoop 1.0 known as MR1, as we can see in Figure 9.

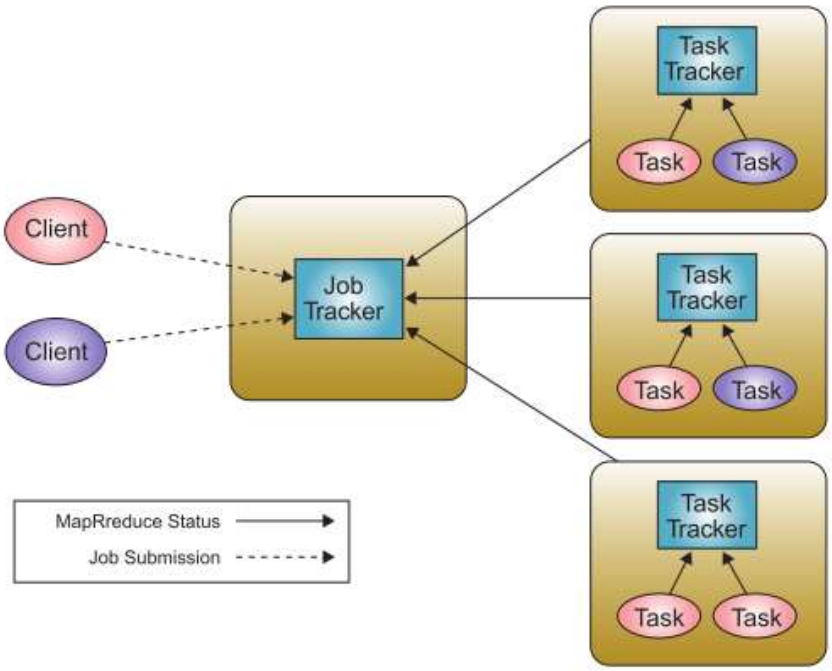


Figure 9. MapReduce (MR1) architecture. Retrieved from [18].

Over the years Hadoop as evolved considerably, in 2013, with the launch of Haddoop 2.0 there was a transition from MapReduce to YARN (Yet Another Resource Negotiator), where YARN rethinks the JobTracker and TaskTracker, replacing them with a ResourceManager, NodeManager and an ApplicationMaster, to solve some problems in MR1, such as scalability on larger clusters [10], concurrent task limits, among other limitations that make a JobTracker a potential choke point [18]. The fundamental idea of YARN is to split up the functionalities of resource management and job scheduling/monitoring into separate daemons.

The ResourceManager is the ultimate authority that arbitrates resources among all the applications in the system, working on scheduling based on resource containers, which specify memory, disk, and CPU.

The NodeManager is the per-machine framework agent who is responsible for containers, monitoring their resource usage (cpu, memory, disk, network) and reporting the same to the ResourceManager/Scheduler.

The ApplicationMaster is, in effect, a framework specific library and is tasked with negotiating resources from the ResourceManager and working with the NodeManager(s) to execute and monitor the tasks.

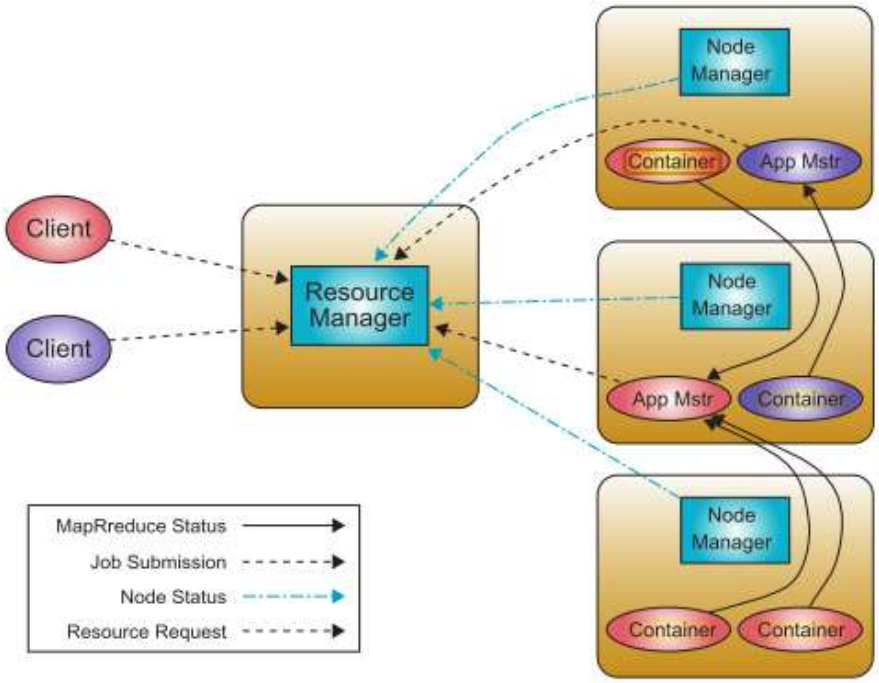


Figure 10. MapReduce v2 architecture. Retrieved from [18].

2.2.3 Hadoop Distributions

To ease the process of configuration and deployment of the Hadoop, it's necessary to do some research regarding Hadoop distributions. Various IT vendors and communities work to improve and enrich Hadoop infrastructure, tools and services, making available proprietary and open source solutions providing a single, integrated offering of all components, pre-tested and certified to work together.

Most Hadoop distributions repackage some core set of packages, including Apache Hadoop itself, along with technologies like Apache Hive. However, the downside is that users may end up with an Hadoop platform composed of various versions of modules from different sources [19].

Between the several distributions available, from our knowledge, the most recognized Hadoop Distributions available in the market are Cloudera, MapR and Hortonworks, being in our opinion Hortonworks and Cloudera the most popular.

Unlike MapR, Hortonworks and Cloudera are open source, where Hortonworks stands out owning an extensive active community that actively participate and help with the problems faced. All the three big players, Cloudera, MapR and Hortonworks use the core Hadoop framework and bundle it for enterprise use [20], also providing downloadable free versions of their distributions. MapR and Cloudera also provide additional premium Hadoop distributions to their paying customers. In a real case scenario, choosing the right Hadoop Distribution for your enterprise is an important decision. Along with the hardware necessary, if we decide to go with a commercial Hadoop distribution it means adding an additional cost. Choosing one over others depends on many factors, including use cases, business needs, Big Data analysis goals, problems to solve, as well as the existing infrastructure.

The authors on [19] recommend to consider the following parameters when choosing among distributions:

- **Technical characteristics:** Hadoop version, available components, proposed functionalities and features (e.g. scalability, data availability, parallel processing, performance, connectivity with the existing application);
- **Convenience level:** easy tools for installation and configuration, user-friendly management interface, possibility to upgrade versions and integrate patches.
- **The maintenance needs**: clusters management, disaster recovery support, and so on.
- **Aditional Costs:** The available budget as well as the cost of the selected solution, including the investments related to the deployment, maintenance and future upgrades and licenses.
- **Integration with existing infrastructure:** Strategies to simplify the solution's integration with the existing infrastructure.

On subChapter 5.2 we will return to this subject, detailing the decisions made to choose between distributions according to our needs and describing our experiences with them.

2.2.3.1 Cloudera Hadoop Distribution (CDH)

CDH (Cloudera Distribution Hadoop) is open-source, Apache Hadoop compatible with distribution provided by Cloudera, being one of the most used Hadoop distributions [5]. Launched in 2009, delivers the core elements of Hadoop, along with centralized administration tool called Cloudera Manager. With Cloudera Manager, the installation process is automated, reducing deployment time from weeks to minutes. After the deployment, it gives users a cluster-wide, real-time view of hosts and services running.

As the most widely deployed Hadoop distribution, CDH is currently running at scale in hundreds of production environments across the largest organizations in banking, telecommunications, media, retail, governments, and more [21]. In addition to that, Cloudera solutions can be integrated to a wide range of existing infrastructure and can handle disparate

workloads and data formats in a single system. Cloudera proposes an easy way for browsing and querying data in Hadoop.

One of the principle exclusive Cloudera modules is Impala (not supported by other Hadoop distributions), an interesting query language module that is compatible with Hadoop that will be described with more detail in Chapter 3.

Cloudera offers two versions of CDH that provide differing levels of cluster and service management capabilities as well as different levels of support. Cloudera Express is completely free to use, on the other hand, Cloudera Enterprise is a proprietary version, which needs to be purchased separately, the final Costs will depend on components and tools adopted.

In order to users get a first contact with CDH, Cloudera provides the latest release of its software as a free download. Cloudera QuickStarts VMs comes in the form of a ready to use virtual machine, with a single-node cluster configured so users can get hands-on CDH, also including tutorials and sample data.

2.2.3.2 Hortonworks Data Platform (HDP)

Hortonworks, founded by Yahoo engineers in 2011, provides a completely open source distribution model for Hadoop, standing as an open and free to use enterprise data platform. Hortonworks Hadoop distribution can easily be downloaded and integrated for use in various applications, provides open source management tools and supports connections with some BI platforms.

The engineers of Hortonworks are also behind most of Hadoop's recent innovations including Yarn, ORC (described in subChapter 5.4) and Tez amongst others, being the first vendor to provide a production ready Hadoop distribution based on Hadoop 2.0.

Though CDH had Hadoop 2.0 features in its earlier versions, all its components were not considered production ready [20]. Like CDH Cloudera manager, HDP also has a tool that helps with creating and operating Hadoop clusters called Ambari. It automates the process of setting up and configuring the Hadoop cluster, after the setup, it keeps an important role in the monitoring and managing of the cluster.

Also like Cloudera, Hortonworks Sandbox is a single node implementation of the Hortonworks Data Platform (HDP). It is packaged as a virtual machine to make evaluation and experimentation with HDP.

2.2.3.3 MapR

The MapR Data Platform is a commercial distribution supporting big data storage and processing through the Apache collection of Hadoop products, as well as its other added components developed by MapR Technologies providing several enterprise-grade proprietary tools to better manage and ensure the resiliency and reliability of data in the Hadoop cluster. It has been enhanced to provide a better reliability, performance and ease of use of Big Data storage, processing and especially analysis with machine learning algorithms [5].

The most significant difference from the previous distributions is that replacement of HDFS with its proprietary file system, MapR-FS, which is designed to provide more efficient

management of data, reliability and ease of use. In contrast, MapR has no NameNode, no handler for the locations of files within the cluster.

Historically, the Namenode has been a single point of failure, and the community has done a great job in trying to resolve that for the purposes of high availability, but like we said previously YARN is one solution to this problem. Metadata for files and directories, all that information is embedded within all the DataNodes spread across the entire cluster. MapR-FS is written in C++, Apache HDFS is written in Java, serving as the company's proprietary implementation of Hadoop Distributed File System that allows applications to concurrently read and write directly to local disk. Hadoop Distributed File System (HDFS), by contrast, has append-only writes and can only read from closed files. Because HDFS is layered over the existing Linux file system, a greater number of input/output (I/O) operations decrease the cluster's performance.

Like the previous distributions, MapR provides a sandbox version that's a self-contained virtual machine, which includes tutorials and demo applications, enabling users to get started quickly with Hadoop. As product's user interface, MapR provides MapR Control System (MCS), which gives Hadoop administrators a single place for configuring, monitoring, and managing their clusters.

2.2.4 NoSQL Databases

Big data is getting bigger and more chaotic every day. This explosion of data is proving to be too large and too complex for relational databases (RDBMS) to handle, and so NoSQL databases (also refered as non-relational databases) have become popular due to the lack of scalability in RDBMS.

This new type of databases NoSQL databases, which are scheme-free, fast, highly scalable, and reliable, began to emerge to handle these data, providing mechanisms for storage and retrieval of data which is modeled in means other than the tabular relations used in relational databases [22]. Since they don't rely on rigid schemas, one of the advantages of using NoSQL is that it allows storage of schema-less data, which makes it well-suited to Big Data environments where the data doesn't have a particular structure, it may be unstructured, like text, documents or videos.

Unlike relational databases, NoSQL databases are not bound by a fixed schema model. Instead of applying schema on write, NoSQL databases apply schema on read. Meaning, instead of coming up with the structure or schema for modeling the data in advance, as is the case with relational databases, consequently NoSQL systems let us store the data as it comes from the datasources, resulting in a more flexible structure.

Although these databases have become popular, we do not suggest that the demise of the tradtional data warehouse is on the horizon. Relational databases will certainly evolve and some organizations (e.g., Facebook) are using mixed database architectures. A recent term is emerging, NewSQL, which combines the relational data model with the benefits of NoSQL systems, such as scalability [10].

The choice between NoSQL and RDBMS is largely dependent on each data needs. If, for example, the main data needs are centered on gathering business intelligence reports or in-depth analytics of large volumes of structured data, then a relational database might be the best fit.

NoSQL is a better choice for businesses whose data workloads are more geared toward the rapid processing and analyzing of vast amounts of varied and unstructured data [10]. There are several of NoSQL solutions available (Voldemort, Redis, Cassandra, MongoDB, CouchDB, HBase among many others), typically divided into four data models, Graph, Key-Value pairs, Columnar and Document databases. For more details on these data models, other characteristics and design of NoSQL databases please refer to [10], [23] and [24].

2.2.5 SQL-on-Hadoop

Hadoop is one of the technologies that is immediately highlighted when discussing Big Data. Although, Hadoop is the wrong choice for interactive queries. As previously referenced, SQL query processing for analytics over Hadoop data has recently gained significant strength, boosting the development of many SQL engines for Hadoop data, being Hive the first native Hadoop querying system.

This development raised the creation of a new concept, SQL-on-Hadoop, a new class of analytical application tools that combine well known SQL-style querying with newer Hadoop data framework elements. The knowledge acquired in the RDBMS is very large, consequently many users are familiar with SQL use. Due to this fact, the number of SQL-on-Hadoop systems have increased significantly, being the solution to Big Data Warehousing and query processing. By supporting established SQL, SQL-on-Hadoop allows a wider range of developers as well as business analysts to be comfortable around the “dreaded” Hadoop [25].

These systems can also play a relevant role at organizational level, since organizations can reduce costs using Big Data tools that supports SQL, reusing employee’s skills without the need to hire specialized personel (like data scientists and business analysts) to process Big Data and extract knowledge.

Tools that support SQL-like querying let business users who already understand SQL apply similar techniques to that data, providing iterative queriying capabilities, where a user asks one question, receives an answer, and then asks another question. In short, SQL-on-Hadoop systems is a class of analytical tools used to provide a SQL interface to Hadoop data framework elements, like HDFS, SQL or NoSQL databases.

The introduction of SQL on Hadoop opens a new door of opportunities as it combines SQL with Hadoop, with the following benefits:

- **Business Continuity:** Existing BI tools most of which use SQL can be integrated with SQL-on- Hadoop tools with relative ease. No major rewrites are required for the existing tools to be productive;

- **Productivity:** Since most analysts can "speak" SQL, enterprises can use their existing human resources and will not need to hire new skilled programmers or retrain their current staff, hence ensuring business continuity;
- **Standard compliance:** SQL has been around for quite some time and as such has matured. Standards are necessary for quality assurance. Also, standard user-friendly interfaces can be made available to the less technically gifted personnel;
- **Interactive queries:** Query execution time in Hadoop is not suitable for ad-hoc interactive queries, however the introduction of an SQL layer improves the response time which is crucial to promote data exploration;
- **Flexibility:** The flexibility that lacked in traditional systems which are intended only for structured data is now available in SQL-on-Hadoop tools. This flexibility is made possible by the use of HDFS as the central repository.

3 Related Work

This chapter aims to describe the existing studies regarding the evaluaton of Big Data querying tools and other providing insights into relevant studies related with our work. Big Data is identified as one of the biggest IT trends of the last few years, which includes a large amount of work regarding querying and processing tools. With this growth, SQL processing, namely SQL-on-Hadoop, has been widely studied, analyzing and evaluating the performance and of several processing tools.

In this context, the work performed on [1] provides a performance comparison of Hive and Impala using the TPC-H benchmark and two TPC-DS inspired workloads, analyzing the I/O efficiency of their columnar formats. The results show that Impala is 3.3X to 4.4X faster than Hive on MapReduce and 2.1X to 2.8X than Hive on Tez for the overall TPC-H experiments. Impala is also 8.2X to 10X faster than Hive on MapReduce and about 4.3X faster than Hive on Tez for the TPC-DS inspired experiments.

The work of [5] recent technologies developed for Big Data, aiming to help to select and adopt the right combination of different Big Data technologies according to their technological needs and specific applications requirements, detailing insights into the architecture, strategies and practices that are currently followed in Big Data computing. In a nutshell, it reviews the main Haddop distributions (Cloudera, HDP and MapR), the Hadoop ecosystem and its main components (like Zookeeper, Hue, HDFS, Hive Oozie, etc), so the reader can select the most suited solutions according to its case. Unlike previous papers, it discusses Big Data technologies by focusing on the four phases of the value chain of big data, i.e., data generation, data acquisition, data storage, and data analysis, presenting for each phase a general background, discussing the technical challenges, and review the latest advances.

In [25], authors claim that Hadoop has limitations that hampered the much needed progress on Big Data analytics, and so, they provide a survey paper to shade more light on SQL-on-Hadoop and highlight their pros and expose the limitations of Hadoop. To surpass these limitations, this work includes a discussion about SQL-on-Hadoop tools like Apache Hive, Cloudera Impala and HAWQ, concluding that this new class of analytical application tools are the future, since they use SQL, the dominant and preferred language for data processing due to its maturity and user-friendliness.

The work performed on [26] focus on Apache Tez, describing their main characteristics and the advantage that it brought on the tools Hive, Spark and Pig, presenting a performance evaluation through a TPC-H derived workload. Results showed that the Tez-based implementation substantially outperforms the traditional MapReduce based one.

In [27] authors evaluate Shark (a component of Spark), Impala and Hive, implementing a micro-benchmarking suite of three classes of SQL queries for both a synthetic and a real world dataset. They find that the different query enginesvary greatly in

performance, Hive is is always being outperformed by the other engines, but whether Impala or Shark is the best performer highly depends on the query type. Results shows that overall Impala is the most CPU efficient, and all query engines have comparable resource consumption for memory, disk and network. Still, authors observe that Impala does not handle large input sizes very well, since when they reach a dataset of 500 GB, Impala presents poor performances.

In [17] the author focus on the performance of the query engine Presto, to see if it meets the requirements to analyse large amounts of data and to conclude if it can be used as a lightweight solution to a data analytics team delivering ad-hoc analysis. To evalutate Presto performance, this work retrieves several performed benchmarks using tools like Hive, Impala, Drill and Spark. The first benchmark presented, performs 11 queries, where Impala and Drill came out as clear winners in execution times, but hey both lacked some functionality that could be found in other solutions such as Hive and Presto. Unfortunatly the authors didn't give any details on which functionalities lack on Impala and Drill.

Another presented benchmark performed by Renmin University, tested five different SQL on Hadoop solutions (Hive, Presto, Shark, Impala and Stinger) since they state Hive is not efficient enough for interactive querying. The results that presented shows that Impala can achieve faster query execution times on most queries. On the other hand, some queries were not able to run on all solutions except Hive, concluding that that none of the solutions were mature enough to be used on a daily basic. Since this work was performed was performed on 2014, this has changed, since these tools evolved since that time. Finally, the author presents a commercial benchmark performed by Pivotal, where they compare their SQL on Hadoop solution HAWQ against Impala, showing results that HAWQ beats Impala in most of the queries. In a nutshell, after analyzing previous results the authors perform they're on benchmark, compaing Hive with Presto, performing 9 queries, overall results showed that Presto is around five times faster than Hive. Presto was proven work best for lighter ad-hoc analysis running on smaller data sets. Presto did not work well with larger data sets and heavier queries containing multiple joins/self joins.

In [28], [29] the motivation for using Hadoop is presented, along with the strengths and limitations of tools like Impala, Hive, BigSQL, HAWQ, and Presto, classifying the SQL on Hadoop engines on three categories (Pure query engine, RDBMS on Hadoop, and Remote query submission solutions) and establishing Cloudera Impala, IBM Big SQL, and Hortonworks Hive (on Tez) as clear winners.

The work of [30] presents the results using TPC-DS queries, comparing response time for a single user and for 10 concurrent users, using Impala, Hive, and Spark, showing that Impala was the only engine that provided interactive query response on both user scenarios. The authors also conclude that Hive is designed for batch processing and ETL, Spark is more oriented to Scala or Java developers, allowing them to embed SQL queries into their Spark proframs. The work of [31] presents a good overview of the reasons why we should use SQL access on Hadoop, also giving an overview of IBM Big SQL and comparing it with Hive, Impala, and HAWQ. This work provides a good insight on SQL-on-Hadoop tools and shows that mainly HAWQ and Impala can provide good performance.

In [32] the authors present Impala giving an overview of its architecture and main components, also demonstrating its performance when compared against other popular SQL-on-Hadoop systems like Spark, Presto, and Hive, also presenting and comparing the compression ratio of popular combinations of file formats like Avro, Parquet and Text. They use the specific file format that performs best on each tool, revealing that Impala has faster response time executing queries in single and multi-user query execution.

In [33], recent trends of storage and computing tools are analyzed, showing their relative capabilities, limitations and environment they are suitable to work with. It is presented a detailed description of four storage tools (HBase, Hive, Neo4j, and Cassandra) and four computing tools (Hadoop MapReduce, Impala, IBM Netezza, and Giraph). The results show that Cloudera Impala, IBM Netezza, and Apache Giraph can achieve very low latency time due to in-memory processing and Hive still does not provide OLTP.

In [34] is studied the integration of Machine Learning in Hadoop ecosystem, studying three different processing paradigms (batch, iterative batch, and real-time streaming) along with a comparison of engines that implement them, like MapReduce, Spark, Flink or Storm. Also, the paper presents a comparison of machine libraries, including Mahout, MLlib and SAMOA concludinf.

In [35] a study on HDFS, MapReduce, Pig, Hive, HBase, and Spark is performed, describing their main characteristics and architectures, showing the cost-effectiveness of Hadoop-based analysis and ease-of-use of the MapReduce technique in parallelization of the many data analysis algorithms. Another benchmark of SQL-like Big Data technologies is presented in [36], which uses queries that involve table scans, aggregations and joins.

When comparing Hive, Presto, Drill, and Spark they conclude that Presto has outstanding runtime on performance over other big data solutions and that SparkSQL has an edge for analytics/machine learning.

The work performed on [37] focus examines the efects of big data analytics on organizations, showing the potential of Big Data analytics in relation to traditional data analytics, introducing diferent techniques which can be used to analyze diferent data sources.

The authors of [38] present a survey of the open source technologies that support big data processing in a real-time/near real-time fashion, including their system architectures and platforms, stating that Hadoop does not effectively accommodate the needs of real-time processing capability. In order to overcome this limitation, they present the open-source real-time processing tools Hadoop Online, S4, Storm, Flume, Spark streaming, Kafka, Scribe, S4, HStreaming and Impala, concluding that the base of real-time processing is the use of in-memory technologies.

In [39] Hadoop Hive and eleven open source tools alternatives are presented, analyzing each one against the needs of Big Data processing, but providing few details about the tools, but stating that all the alternatives surpass Hive in terms of performance. Still they perform a good comparison od SQL on Hadoop tools according to user needs. In [40] characteristics like latency and ANSI SQL completeness, are used to evaluate Drill, Hive, Impala, and SparkSQL. It is highlighted that Hive has low maintenance and is simple to

learn, but not suitable for real-time queries, Impala has lower query latency, but memory errors are very frequent.

In [41], the authors use a denormalized TPC-H schema testing it in a low cost cluster with Hive, Spark, Presto, and Drill. The results show that it's possible to achieve adequate query execution times on modest hardware, demonstrating that Presto has advantage over the other evaluated options, being suited for interactive queries.

Although a comprehensive set of works in this field is already available, most of them are focus on extensive and sometimes confusing theoretical research, in some cases showing practical approaches and benchmarks in order to demonstrate tools performance. Also, we felt a light confusion since previous works usually approach a several number of work frameworks and tools, reads don't quite understand what components are used for what ends.

Despite the existence of some work in the Big Data querying and processing, there is a lack of studies, analyzing scalability evaluation and comparing query execution time, specifying the ideal scenario to use the tools in a clear way, most of the studies have deep technical detail on several tools, consequently, like we refered previously, users with less experience in this context can feel confused, contributing to the complexity of concepts and user rejection to work in this environment.

In our work, we focus on the selection the state of the art Big Data processing tools, describing their architecture in a simple manner so that any reader can understand the concepts. We cover from a more academic to experimental approach, presenting essential concepts related to Big Data and Hadoop Environment, theoretical description and characterization of Big Data querying tools, also describing how we deployed and use them in a configured environment (Hadoop Cluster), close to a real case scenario. From our knowledge, the previously performed studies don't compare directly the set of state of the art tools what we chose to analyze. Besides the experimental evaluation of Drill, HAWQ, Hive, Presto, and Spark, we also test Impala, which is very popular between users, but not compared in the previous works with all the tools referenced, maybe due to the fact that it requires the deployment of distinct Hadoop distributions as we will see ahead in this work.

Regarding the deployment of Hadoop distributions, we also describe briefly the steps followed to prepare a stable, close to reality scenatio ready for the deployment of Hadoop and installation of the selected tools, also specifying how to use them.

In our work, we install Impala and compare it against the other tools, using the most optimized file formats available for each tool, comparing Impala storing data using Parquet files and the remaining tools with the Stinger initiative ORC file format, performing a series of performance experiments, to conclude in which scenarios, they are suitable or ideal and those where is not recommended to use them.

4 Big Data Querying Tools

The use of highly distributed and scalable systems to process Big Data is considered one of the recent key technological developments. Occasionally, users do not know queries in advance and need to execute *ad hoc* queries within seconds, even at scale. In particular, the community claims that Hadoop is the wrong choice for interactive queries that have a target response time of a few seconds or milliseconds [19].

To overcome this limitation and perform *ad hoc* queries on huge datasets there are several Big Data querying tools, running on top of Hadoop ecosystem, which together with the concept of parallel processing provide fast interactive queries. Not all SQL processing tools are equal, being a challenge picking the right tool. Besides querying, some of these tools provide components that can support Big Data analytics.

To query data, some of these systems proposed proprietary query languages or application program interfaces, while others have recognized the benefits of using SQL, originating the term SQL-on-Hadoop. Thus, this concept opens the Big Data World to a wider audience, there is a need to identify clearly the SQL-on-Hadoop options. Since they run on top of Hadoop, and this ecosystem has a wide number of components and technologies, we noticed that sometimes it is difficult to identify which tools are dedicated to query and analyze Big Data.

This Chapter describes in detail a selected set of Big Data querying tools, in our opinion this set represents the best options available [25], [41] and [32]. All the query engines have limitations and are not as robust and mature as the standard SQL query tools that are available on traditional warehouses. For comparison and due to their popularity, we analyze in this Chapter Drill, HAWQ, Hive, Impala, Presto, and Spark.

4.1 Apache Drill

Drill is an open-source distributed system that supports data-intensive distributed applications for interactive *ad hoc* analysis of large-scale datasets. It was developed with the goal of providing low latency and faster interactive queries, supporting several data storage NoSQL databases like the Hadoop Distributed File System (HDFS), Hive or HBase and ANSI SQL to query data, being capable of perform mathematical and statistical functions, string and dates manipulation. To access this variety of data sources, Drill provides storage plugin interfaces to read from and write to data sources, defining a set of optimization rules to help with efficient and faster execution of Drill queries on a specific data source.

Besides the mentioned data storages, data can reside in different file formats like CSV, TSV, JSON, PARQUET and AVRO. Being simple to install on a cluster and using distributed cache to manage metadata, it can scale-up to a very large cluster with thousands of

nodes. On top of all these characteristics, it offers connectivity and compatibility with BI tools like Tableau, Microstrategy, Qlikview and Tibco [36].

As a core component, it has the Drillbit service, which is responsible for accepting requests from the client, processing the queries, and returning results to the client. Drillbits can be installed and run on all nodes in a Hadoop cluster, forming a distributed cluster environment, also meaning that queries can be submitted through any node in the cluster.

To easily add and configure these connectors, check if drillbits are up and query execution logs, Apache Drill provides a WEB UI accessible by default on port 8047 of the Drillbit nodes. When a client issues a query to Drill, the execution process typically consists in contact Zookeeper to return the available Drillbits in the cluster to which the query can be submitted. ZooKeeper maintains cluster membership, health-check information and identifies the appropriate nodes to execute query fragments. Drill architecture is shown in Figure 11.

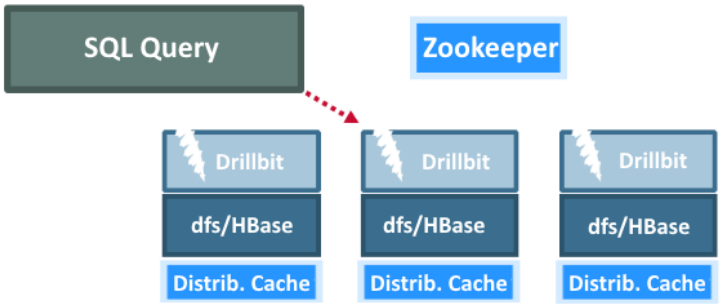


Figure 11. Drill architecture. Adapted From [42].

The Drillbits are composed by components like:

1. RPC (Remote Procedure Call) end-point, allowing communication with the clients and receiving queries through the RPC protocol;
2. SQLParser, optimizing queries and generating a distributed query plan that is optimized for fast and efficient execution. To perform these optimizations, Drill uses Calcite, an open source SQL parser framework, to parse incoming queries.
3. Optimizer, responsible for managing standard database optimizations, providing a distributed query plan for efficient execution across different nodes (see Figure 12).

In short, Drill provides interactive query capabilities using its own SQL execution engine, enabling traditional Business Intelligence and analytics from different sources, suited for advanced analytics workflows, offering query execution time that vary between milliseconds to minutes depending on the query complexity and the size of the dataset [43].

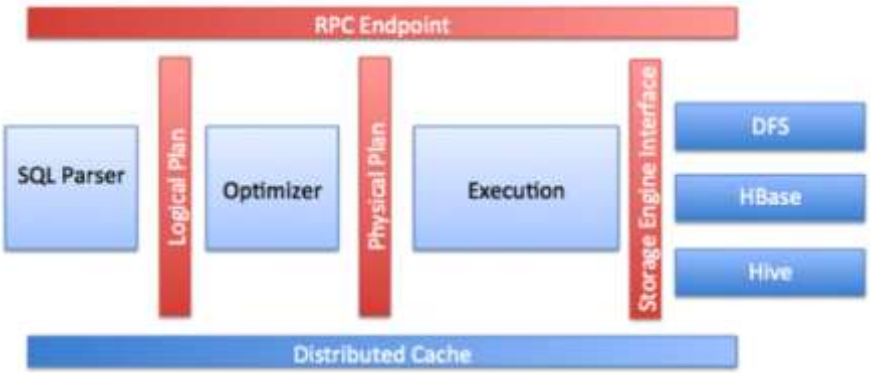


Figure 12. Drillbit Components Adapted From [42].

4.2 HAWQ

HAWQ, Hadoop With Query, recently brought into the Apache Foundation from the well-established Pivotal HAWQ, is a Hadoop native SQL query engine that combines the key technological advantages of Massive Parallel Processing (MPP) with the scalability and convenience of Hadoop, delivering industry-leading performance [25].

This parallel SQL query engine built on top of the HDFS, claims to be the world's fastest SQL engine on Hadoop [44]. It adopts a layered architecture and relies on HDFS for data replication and fault tolerance. HAWQ relies on both the PostgreSQL database and the HDFS storage as its backend storage mechanism, meaning that HAWQ can support the full ANSI SQL syntax. Another capability of HAWQ is its integration with MADlib, providing machine-learning capabilities directly in SQL.

Along with the Pivotal Extension Framework (PXF), HAWQ can work with data from several data sources like HBase, Hive, Text, Avro and Parquet, being able to run and be managed on Hortonworks HDP. The architecture of the HAWQ engine (see Figure 13) includes the following main components [31]:

1. **HAWQ master:** entry point responsible for accepting the connections from the clients and manages the system tables that contain metadata information about HAWQ itself, being also responsible for parsing and optimizing the queries and generating the query execution plan;
2. **HAWQ segments:** represents the processing units, responsible for running the local database operations on their own data sets;
3. **HAWQ storage nodes:** used for storing all the user data (HAWQ relies on a proprietary file format for storing the HDFS data);
4. **HAWQ interconnect:** Responsible for managing the inter-process communication between segments during query execution.

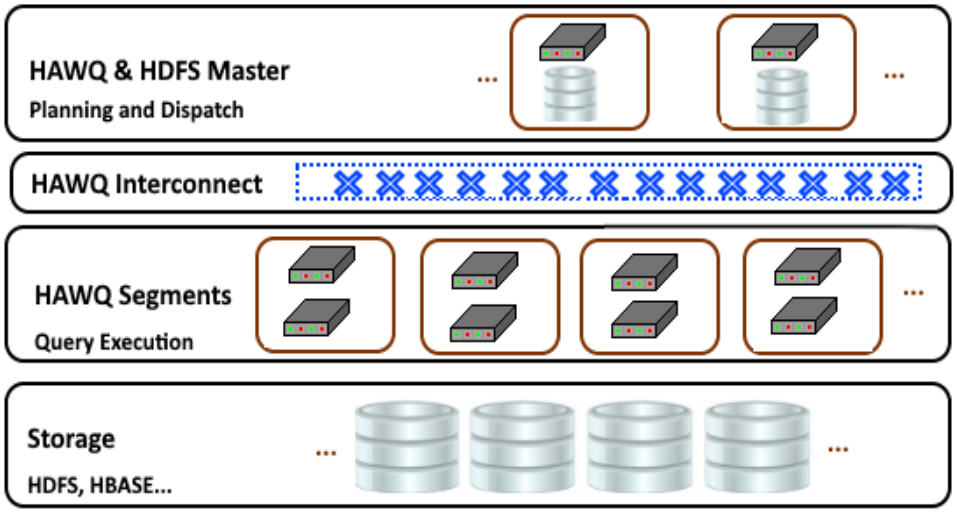


Figure 13. HAWQ Architecture. Adapted From [45].

Comparing with the other presented tools, it is easy to say that HAWQ has the most elaborate architecture. Queries are executed via PostgreSQL client and are sent to a HAWQ master, there they are parsed, optimized, fragmented and dispatched to HAWQ segments across the nodes for execution. During query execution YARN takes care of all resource management matters for overall performance and cluster stability.

For the high availability of the master node, a standby master instance can be optionally deployed on a separate host (like HDFS Namenode and Secondary Namenode). The standby master host serves as backup when the primary master host becomes unavailable. HAWQ Master also has a fault detector that checks the health of all segments periodically.

4.3 Hive

Hive is a system that supports the processing and analysis of data stored in Hadoop, more specifically in Hadoop Distributed File System (HDFS). We have referred this tool previously as one of the Hadoop's environment components, considered the data warehouse storage system for Big Data and the first to support SQL-on-Hadoop, using an underlying framework such as MapReduce (or more recently, Tez) to process SQL-like statements. It has been used by many organizations, such as Amazon, to store and process large amounts of data.

Being frequently considered as a high-latency system, oriented to batch workloads instead of interactive querying, Hive has been target of constant development to improve its performance.

Out of all these improvements we highlight the Stinger initiative. Stinger initiative was implement by Hortonworks, introducing ORC, a columnar format providing high compression and high performance (detailed in 5.4) and the execution engine Tez that optimizes Hive job execution which aims to Hive's latency caused by MapReduce. Built on the Hadoop platform, it supports familiar relational database concepts such as tables, columns, and partitions and includes some SQL support for unstructured data [46]. Hive

gives structure to data and performs *ad hoc* querying and analysis using HiveQL, a SQL-like query language.

To provide a schema reading functionality and semantics check on queries, Hive stores metadata in Hive Metastore, using this information, Hive Server transforms the HiveQL into MapReduce or Tez jobs. Hive Metastore has become a standard location to tabular data, it exposes the hive metadata to other tools through Hcatalog (a table and storage management layer for Hadoop that enables users with different data processing tools to read and write data), like Apache Drill. Also like Drill, Hive is powered by the query optimizer and execution framework Calcite, complementing Hive with the (CBO) component.

The main goal of a CBO is to generate efficient execution plans by examining the tables and conditions specified in the query, ultimately cutting down in query execution time and reducing resource utilization.

The work of [47] shows clearly CBO performance improvements, running several queries with CBO on and off. We have realized that Hive is in constant development in order to improve performance, especially by Hortonworks, distributors of the already mentioned HDP Hadoop Distribution. As part of HDP 2.5, Hortonworks introduced as a technical preview, Hive LLAP (Low Latency Analytical Processing), an optional daemon that leverages Tez, delivering the promess of low latency, concurrency and overall performance that were not possible with earlier versions of HDP.

Hive LLAP is being developed to give Hive a new architecture that delivers MPP performance at Hadoop scale through a combination of optimized in-memory caching and persistent query executors that scale elastically within YARN clusters. Hive supports all the common primitive data formats such as BIGINT, BINARY, BOOLEAN, CHAR, DECIMAL, DOUBLE, FLOAT, INT, SMALLINT, STRING, TIMESTAMP, and TINYINT. In addition, analysts can combine primitive data types to form complex data types, such as structs, maps and arrays.

On the other hand, HiveQL the query language of Apache Hive also has its own limitations, since some SQL features not yet available such as update and delete queries, and also, several restrictions on the use of subqueries. Transactions, are also not supported, therefore, it cannot be used as OLTP tool [48]. In terms of processing, SQL queries are submitted to Hive and executed through several Hive components, represented in Figure 14, as follows:

1. Hive compiles the query;
2. An execution engine, like Tez or MapReduce, executes the compiled query;
3. The resource manager, YARN, allocates resources for applications across the Hadoop cluster;
4. Data manipulated in the query is stored in HDFS;
5. Query results are made available over a JDBC/ODBC connection.

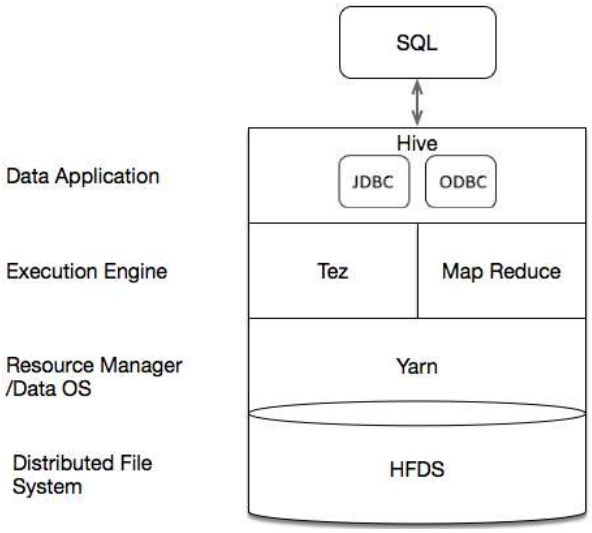


Figure 14. Hive Architecture. Based on [49].

Summarizing, Hive has a good interface for anyone from the relational database world, demanding low maintenance and being simple to learn. We can think on Hive as a standard, it was the first to support SQL-on-Hadoop and it is included in all Hadoop distributions.

In the first distributions, Hive relied on Hadoop's MapReduce suffering from poor performance, being more appropriate for large scans, where query performance may not be so critical. It was not appropriate for On-Line Analytical Processing (OLTP) tasks. It is worth mentioning that using MapReduce is the main criticisms made to Hive, as the conversion to MapReduce jobs leads to higher query latency. Later versions of Hive can run on Tez, a tool that aims to enhance performance and allied with CBO delivers Hive interactivity with improved query performance and latency.

4.4 Impala

Impala is an open-source, state-of-the-art Massive Parallel Processing (MPP) SQL query engine designed for performance, real time, low latency and high concurrency processing. It was developed in C++ and Java, with the objective of combining the SQL support and multi-user performance of a traditional analytical database with the scalability and flexibility of Apache Hadoop [25]. Impala implements a distributed architecture that can run on hundreds of machines in an existing Hadoop cluster (see Figure 15).

This tool can use two storage systems, HDFS or HBase, being possible to query data in both. It can also be integrated with Business Intelligence tools like Tableau, Pentaho, Micro Strategy and Zoom Data. Impala has been built to extend the key components of Hive, e.g., SQL syntax (HiveQL, meaning that like Hive it supports relatively little DML, there is no UPDATE or DELETE statements), metadata and schemas [50].

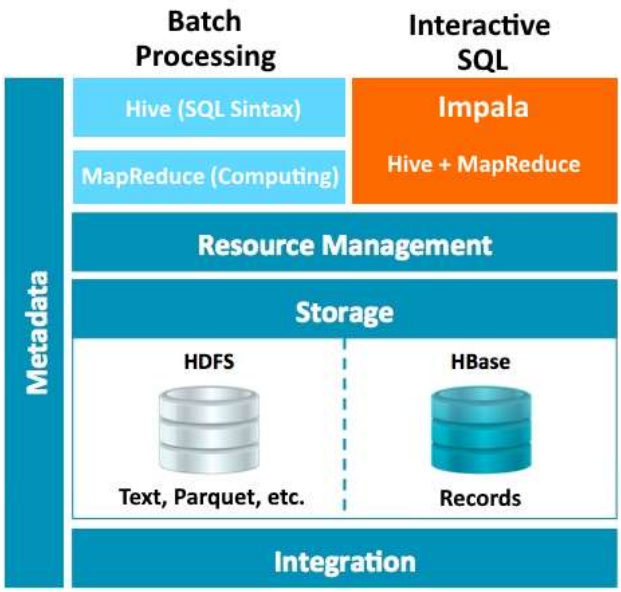


Figure 15. Impala architecture. Adapted from [50].

Impala relies on in-memory join implementations, meaning that queries can fail if the joined tables cannot fit into memory [46].

This is a major drawback, but if we consider that real-life Big Data processing scenarios are built on top of Hadoop clusters, containing high amounts of RAM memory, this drawback may never exist. If the amount of memory available ensures that data that is being processed by queries fits, in-memory processing can provide outstanding performances.

Other drawbacks regarding the utilization of this include the lack of support for serialization and deserialization (if records or files are added to the data directory in HDFS, the tables need to be refreshed) and lack of fault tolerance, if a node fails in the middle of processing, the whole query must be restarted.

In summary, Impala has a large advantage since processing is done in-memory, meaning that Impala does not materialize intermediate results to disks, consequently reducing latency and Disk IO, especially in real-time and *ad hoc* queries, as long as the runtime is short enough that node failures during the query execution are unlikely and the data involved in the query fits in memory.

4.5 Presto

Presto is an open source distributed SQL query engine for running interactive analytical queries against data sources of all sizes ranging from gigabytes to petabytes, targeted at analysts who expect response times ranging from sub-second to minutes. It was developed by Facebook, making available to give Hadoop some SQL-like capabilities, being optimized for low latency and interactive query analysis.

Facebook uses Presto for interactive queries integrating several internal data stores, including its 300PB data warehouse. Over 1,000 Facebook employees use Presto daily to run more than 30,000 queries, each scanning over a petabyte per day [50]. Presto supports several

SQL features including joins, aggregations, and subqueries, as well as other features that include JSON, URL functions, strings and regular expression functions.

A single Presto query can combine data from multiple sources, allowing for analytics across several data sources in the organization, meaning that it was not only designed for querying data present in HDFS but also in other data sources, including relational or non-relational databases.

Using in-memory processing, instead of MapReduce, it avoids side steps, unnecessary I/O and latency, allowing faster response times. As a result, Facebook claims that Presto runs 10 times faster than Hive [36]. Presto can perform simple queries in few hundred milliseconds and more complex in few minutes.

Running on a cluster of machines, it operates on an architecture (illustrated in Figure 16) that includes a coordinator, multiple workers and a client. The client Presto CLI, is a terminal-based interactive shell that submits SQL statements to a coordinator to get the result. Presto Coordinator, Parses the SQL queries for defining the query execution plan. Finally, Presto Workers, receive assignments from the coordinator and delivering results.

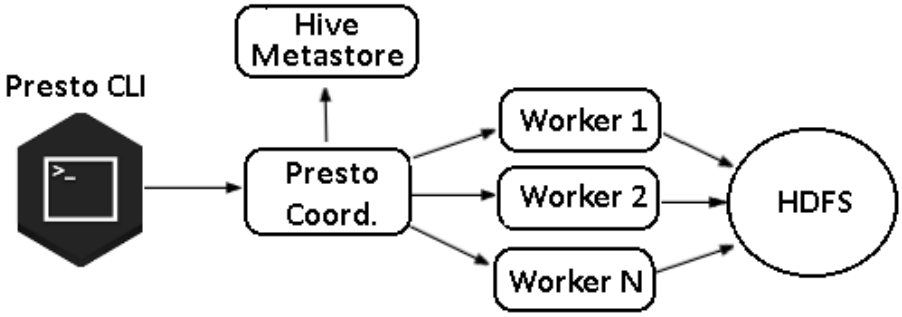


Figure 16. Presto architecture. Adapted from [51].

One of the main drawbacks is the limitation on the maximum amount of memory each query can have, so if a query requires a large amount of memory it will simply fail, similarly to what we described for Impala. In order to monitor and manage queries, Presto also provides a web interface, giving several information to the user about how much workers are up, recently completed or failed queries, and query memory consumption in real-time.

The web interface is accessible on the Presto coordinator via HTTP, using the HTTP port number specified in the coordinator configuration properties.

4.6 Spark

Spark is a highly distributed processing framework that provides an ease of use tool for efficient analytics on heterogeneous data. It was originally developed in 2009 in UC Berkeley's AMPLab, and open sourced in 2010 as an Apache project, running on top of Hadoop [35]. Its main characteristics are the way it works, it uses in-memory cluster computing that increases the process speed of an application. Spark is like Hadoop but it is based on in- memory system to improve performance [19].

Spark provides full access and compatibility with Hive existing data and uses the concept of Resilient Distributed Datasets (RDD), which describes an immutable collection of objects that are partitioned and distributed across multiple nodes of a cluster, allowing parallel processing. One of the key modules of Spark is Spark SQL, used for processing structured data with DataFrames (equivalent to relational database tables) that organize data in named columns. The advantage of using Spark SQL-on-Hadoop tool over other tools, is the ease of using Machine Learning with SQL. To extend the vocabulary of SparkSQL, it is possible to plug-in custom user defined functions (UDFs).

Spark uses a master/worker architecture, where a single coordinator, the master, manages all workers, executing tasks, as depicted in Figure 17.

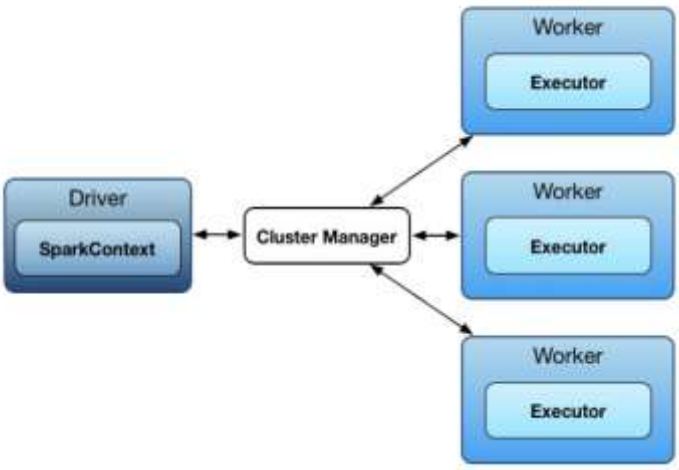


Figure 17. Spark architecture. Retrieved from [52].

Spark is designed for supporting a large range of workloads like batch applications, interactive queries, iterative algorithms and streaming, through the following components:

- Spark Core, the underlyng general execution engine for Spark platform that all other
- functionality is built upon.
- Spark Streaming, leveraging fast scheduling capability to perform streaming analytics with Spark Core;
- MLLIB (Machine Learning Library), a distributed machine learning framework above Spark, which is nine times faster than Hadoop disk-based version of Apache Mahout [35];
- GraphX, a distributed graph-processing framework on top of Spark for simplifying analytical tasks.

These components allow real-time analysis, discovery and processing patterns, complex math, statistics, or machine learning, being a good choice in real time processing scenarios, e.g. event detection.

As referenced, on of Spark components its Spark Streaming. This component was included in latest versions of Spark, an extension of the core Spark API for the processing of data streams. Spark Streaming can consume static and streaming data from various sources, process data using Spark SQL and then apply machine learning techniques from MLlib. This is extremely useful when we remember the concept of IoT. Sensors and other IoT devices, constantly generating data that needs to be monitored and acted upon quickly. As a result, the need for large scale, real-time stream processing is more evident than ever before.

In [53], the authors describe how we could detect an earthquake by analyzing Twitter real-time data, showing that this technique is able to inform about an earthquake in Japan quicker than the Japan Meteorological agency.

4.7 Picking the Right Tool

As previously mentioned, Big Data is frequently unstructured, with different formats and requiring the integration of several sources and performance is the priority on SQL-on-Hadoop tools, so efficient querying processing structure is demanded. In previous chapters we reference the tools that from our point of view stand out from others.

To determine the best querying tools, we need to know not only the existent state of the art technologies but also what requirements and characteristics they should fulfill to successfully store and process Big Data efficiently. Based on the works of [43], [54] and [39] the following characteristics can be defined as the main requirements for an efficient Big Data processing architecture:

- **Scalability:** Scaling is the ability of the system to adapt to increased demands in terms of data processing; linear scalability is necessary for the explosive growth of data size;
- **High throughput (processing speed):** Big Data's velocity demands that data be ingested and processed at high speeds, this requiring an infrastructure that is extremely fast across input/output (I/O), processing, and storage;
- **High degree of parallelism:** By processing data in parallel, we can distribute the load across multiple machines, each one having its own copy of the data, but processing a different part;
- **Programming language support:** The support of programming languages can be useful for integrating different BI tools or for supporting other developments;
- **SQL Support:** Since SQL is the base for queries processing;
- **Distributed Architecture:** Distributed processing across several servers (nodes), providing parallel computing;

- **Fault Tolerance:** Mechanisms for detecting failures and for recovering from them;
- **Single Point of Failure:** Identifying when the whole query is aborted if one of the processing nodes fails;
- **Machine Learning Algorithms/Tools:** For advanced data processing, allowing the identification of hidden patterns and trends in Big Data.

Considering the previously analyzed state of the art querying engines and their core features presented,
Table 2 shows the key evaluation characteristics for comparing these tools.

Table 2. Big Data analytical tools comparison.

Feature	***Drill***	***HAWQ***	***Hive***	***Impala***	***Presto***	***Spark***
Owner	Community	Greenplum	Community	Cloudera	Facebook	Community
Cluster Size Limit(node)	Thousands	Thousands	Hundreds	Thousands	Thousands	Thousands
SQL Support	ANSI SQL	ANSI SQL	HiveQL	HiveQL	ANSI SQL	Spark SQL (HiveQL and UDF's)
Latency	Low	Low	Medium	Low	Low	Low
Data Size Limit	Gigabytes to Petabytes		Terabytes	Gigabytes to Petabytes	Gigabytes to Petabytes	Terabytes
Scalability	High	High	High	Very High	Very High	High
Machine Learning	No	Yes	No	Yes	No	Yes
Data Visualization	No Native support, compatibility with BI tools	No Native support, compatibility with BI tools	No Native support, compatibility with BI tools	No Native support, compatibility with BI tools	No Native support, compatibility with BI tools	No Native support, compatibility with BI tools
Processing Speed	Fast	Very Fast	Slow	Very Fast	Very Fast	Fast
Language Support	C++, Java and other languages JDBC/ ODBC	Python, Perl, Java, C/C++, R	Java	All languages supporting JDBC/ ODBC	C, Java, Node.js, PHP, Python, R, Ruby	Java, Python, R, Scala, JDBC/ ODBC
Use Cases	Real-time Interactive queries and analysis/BI	Batch processing and interactive queries	Batch processing	Real-time Interactive queries and analysis/BI	Real-time Interactive queries	Batch and stream processing, interactive queries and ML
Distributed Architecture	Yes	Yes	Yes	Yes	Yes	Yes
Single Point of Failure in Query Execution	No	Yes	Yes, if failure at the master node.	Yes, if any host quits query execution	Yes, if any host quits query execution	No
Fault Tolerance	Yes	Yes	Yes	No	Yes	Yes
File/Storage Formats	CSV, TSV, PSV, Parquet, Hadoop Sequence Files (Key-	Text, Parquet, Avro, CSV,ORC	ORC, AVRO, Parquet, and Text	Parquet, Text, Avro, RCFile or Sequence files	Text, Sequence Files, RCFile, ORC and Parquet	JSON, Parquet, CSV, ORC, RDDs, Hive Tables and External

	value pairs), ORC					Databases

From all the referenced evaluation features, in terms of processing performance we consider the most relevant ones the processing speed and/or latency, since they are directly linked. As we analyzed the tools, we became aware that the options that use in-memory processing instead of MapReduce are better performers, since all processing is made in-memory the associated latency of MapReduce's operations does not exist, consequently resulting in quicker response times. Impala, Presto and HAWQ are examples of in-memory processing tools and if we consider

Table 2, we can see that they are associated to low latency and very fast processing.

Comparing in terms of SQL compliance, Drill, HAWQ and Presto have a more extensive SQL compatibility. Spark uses Spark SQL that follows Hive style HiveQL sintax. Impala and Hive, both use HiveQL but Hive is more appropriate for long-running batch processing, offering robustness and low maintenance.

Impala provides a better support for real-time analytic/*ad hoc* queries having less latency than Hive since it bypasses MapReduce. Using Hive with Tez instead of MapReduce makes Hive a possible solution for real-time analytic/*ad hoc* queries and OLTP environments, but based on some benchmarks performed previously (e.g [1], [32]), Impala can still provide better response times. Presto was developed to address the ad hoc interactive use cases, it is used by organizations for data exploration (e.g Netflix), providing better performance than Hive, Drill and Spark [55]. Still, Cloudera claims that Impala is 5.3 to 7.5 times faster than Presto [29]. Spark integrates a machine learning library (MLib), which makes it suitable for analytics, being an adequate choice when the objective is not just querying data but working with it in an exploratory manner, being the only tool to support streaming analysis. Spark has short query response times, being faster than Hive, even so, Impala seems to provide faster response times [30] [32]. Of all the presented tools, after the analysis of the state of the art, HAWQ is the less known option. However, it can provide strong processing with full ANSI SQL support and provides machine learning and data mining algorithms. On several benchmarks, HAWQ managed to achieve performance improvements. On [56] we see that comparing with batch-oriented queries running against a Hadoop cluster 10x to 600x, improvements that could turn batch systems into interactive ones.

5 Experimental Setup and Methodology

In previous chapters, we introduced the problem outline, with the theoretical background, so that even users with few experience in this area can understand all the concepts and even use it as guide. In this chapter, we explain the approach followed to carry out the pratical experimentation, describing the implementation environment, software and hardware configuration and decisions made to perform the benchmarks.

We must say it took a long time to find an appropriate infrastructure to host an Hadoop cluster, since it's not viable to run this kind of environments (Hadoop) on a single machine, even though we tried through the configuration of several virtual machines running on our personal laptop, to simulate a cluster with several nodes, but naturally a single pc couldn't perform properly under these circumstances.

After we gathered the required clusters infrastructures we started to deploy the necessary software to run a stable Hadoop Environment, to support the installation and practical performance analysis of the previous analyzed tools. In this chapter, we describe the decisions made to deploy those tools and how we will evaluate their performance using the TPC benchmarks TPC-H and TPC-DS.

5.1 Implementation Environment

The first contact with this kind of tools was made using only a personal laptop, which was unfeasible, since Big Data applications stress all system components, such as CPU cores, memory, storage and network I/O, too heavy for a single machine. Remember, the Hadoop ecosystem is designed with a parallel environment in mind. To support distributed parallel processing and boost the speed of data analysis applications we use a cluster with 4 nodes, kindly ceded by INCD (Infraestrutura Nacional de Computação Distribuída). INCD is an entity that provides computation services and storage to the scientific community in all domains supported by FCT (Fundação de Ciência e Tecnologia).

In order to run a distributed Hadoop environment in our Hadoop cluster, the four nodes should follow the master/worker architecture referenced on previous chapters. Considering that the master node should be able to connerct to all worker nodes SSH (without requiring a password, so key based authentication), such that they can communicate without any prompt for password.

To access the clusters resources, we need a bridge/access from our personal laptop. The entrance point to the cluster was the Master node, and so, we configured a public IP on the target machine, so we could make a remote SSH connection from our personal laptop (Putty was used as an SSH client to perform the remote connections).

To grant access to that remote machine, we had to generate our personal computers private SSH key (through PuttyGen, installed along with PuTTY, it generates public and

private SSH key pairs), and authorize it on the master node, to gain access to the cluster from our personal computer.

After we access the master node we gain access to all the other nodes since they were configured internally to be able communicate between them (through the distribution of each nodes SSH keys). The described architecture is illustrated in Figure 18.

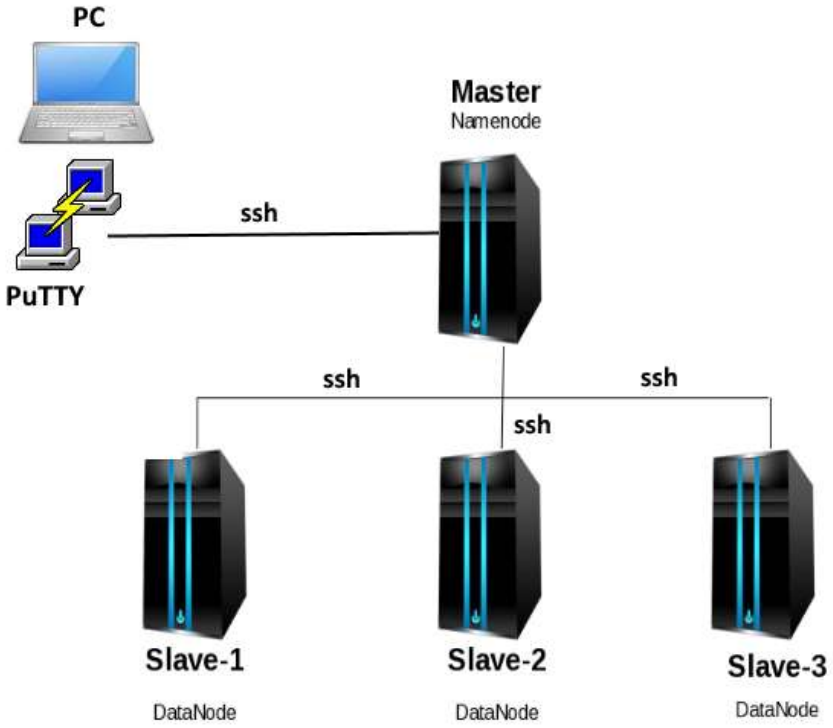


Figure 18. Cluster access architecture.

In short, the hardware configuration of each node includes one octa-core CPU 1.80GHz, 16 GB of RAM and one SATA disk with sizes ranging between 120 and 240 gigabytes, connected through gigabit Ethernet (1000 megabits per second) to achieve the gigabit data rate, with 64-bit CentOS Linux 7 as the operating system.

As seen frequently in the literature, SQL-on-Hadoop systems need significant amounts of RAM to process data, benefiting from the availability of a higher number of nodes available (e.g. the cluster in [1] use 21 nodes and 96 GB of RAM and in [32] has 10 nodes and 48GB of RAM). Even though, as we see in [41], it is possible to perform this kind of experiences on low cost clusters.

5.2 Choosing a Distribution

After we have our cluster up and running with all the conditions to receive the Hadoop ecosystem, we must consider an Hadoop distribution to install. In subchapter 2.2.3 we described what we consider the most popular Hadoop distributions, and now, according to the tools that we pretend to analyze we must choose which ones are applicable. From the beginning of this work, Hortonworks was noticed as reference of Hadoop distributions,

therefore, from an early stage we started to consider it as a very viable option to deploy on our cluster.

Since our works focus is on Big Data querying tools we must choose a distribution that supports all our set of tools. During the theoretical investigation of the Big Data querying tools, we realized that it wasn't possible to have a distribution that supported all the analyzed tools. Hortonworks, our preferred distribution due to its popularity and extense community, supports all major tools and services in the Apache Hadoop ecosystem, including the Big Data querying tools analyzed previously except Cloudera Impala, which requires exclusively on Cloudera's Hadoop distribution CDH.

From our knowledge, it would also be possible to deploy Presto along with CDH, for Drill we are not so sure, since the information on Drill official website (https://drill.apache.org/) claims that it can be deployed along all Hadoop distributions, but we have seen several users reporting problems. Regarding Apache HAWQ, there is the option of installing it in HDP through an Ambari plug-in, being that we couldn't find any information about deploying HAWQ on CDH, therefore we assumed its not possible to deploy this tool on Cloudera's distribution.

Resuming we decide to test the Drill, Hive, Impala, HAWQ and Presto with HDP and CDH only for Impala. Since Hortonworks has such impact in the Hadoop scope and Cloudera covers the remaining tool that wasn't supported in HDP, MapR wasn't consider.

We must point that the Impala's mandatory use of CDH (although other Hadoop distributions include Impala, like MapR, though it is not tested or supported by the official vendor Cloudera), constitutes a major drawback, considering we installed HDP first, to test Impala, we must clear all HDP content of all nodes in the cluster and then deploy CDH.

If we have CDH deployed and we want to go back to HDP (or to test a tool that only runs with HDP), we must do the same thing, uninstall and deploy all over again, a time-consuming process.

Although the main scope of this work isn't comparing Hadoop's distributions, but choose the ones that fit our purposes, although since the set of querying tools referenced previously always run on top of Hadoop, we consider relevant to describe them and present tool compatibility and also describe briefly the experience we had when deploying them in our cluster.

5.2.1 Hortonworks Data Platform 2.5

HDP 2.5 was quickly deployed in our cluster thanks to Ambari, which allows easy installation and secure configuration of HDP, since it automates the process of setting up and configuring of all essential components. To deploy HDP using Ambari, we must install two Ambari components, Ambari Server (in the master nodes) and Ambari Agent (in all nodes), in which the server will communicate with the agents to install HDP across all nodes.

In a later stage the Ambari agents will be responsible for updating the status of every node with the help of various operational metrics. HDP supports all major tools and services in the Apache Hadoop ecosystem, including Hive, and Spark, two of the tools that we

propose to study. Some of these tools are client programs only, others require daemon services to be installed and configured on all or some nodes of the cluster. Some even have "Master" daemons that control cluster-wide operation, like HDFS NameNode. We also know that, as referenced in 3.3, Hortonworks is continuily developing Hive, this version, includes a technical preview of LLAP, but since its only a technical preview we didn't activate this functionality when we ran the queries in Hive.

One of the challenges we faced deploying this distribution was how to distribute all daemons sensibly across the cluster as to avoid overcharging the nodes. For this purpose, Ambari provides a default configuration that can be tailored according to the cluster characteristics. A second important aspect of Ambari is the capacity to track cluster usage and health after the initial setup has been completed.

Most services in the plain Hadoop ecosystem host simple HTML status pages on each node on which they run, but these status pages are all independent of each other, meaning we had access several URL's, with port numbers that are hard to remember.

Luckily, Ambari provides an intuitive Web UI (by default available on port 8080 of the Ambari server node) that includes a detailed dashboard (see Figure 19) through which it makes aggregated statistics for all the services available. This overview is immensely useful to monitor the load and resources of our cluster, otherwise a complete picture is not available.

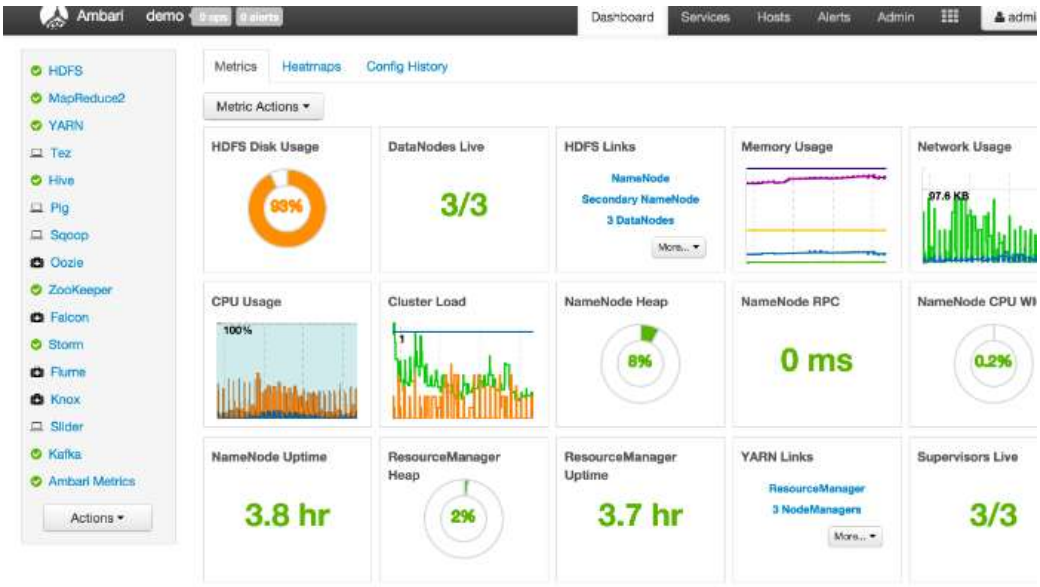


Figure 19. Ambari dashboard.

Apache Ambari also includes the Ambari Views Framework, which enables developers to create UI components, or Views, that "plug into" the Ambari Web interface. When we deployed HDP through Ambari, it automatically created some instances of Views, depending on the services chosen. For example, if Apache YARN service is added to the cluster, the YARN Queue Manager View displays to Ambari web users. Consequently, when we deployed HDP in our cluster, views were created, according to the services that we chose. Figure 20 shows the automatically created views.

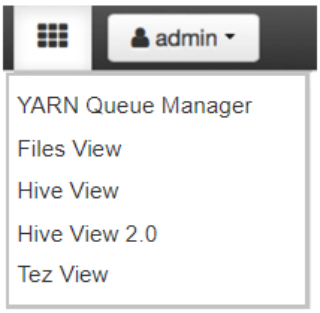


Figure 20. Ambari Views.

Using Views enables to extend and customize the Ambari web to meet our specific needs. In our case we find Files and Hive views extremely useful. Files view provides a convenient way to access files and folders in cluster's file system, using a browser-style user interface. Hive view is designed to help you optimize, and execute Hive queries, allowing to browse databases, write queries or browse query results, manage query execution jobs and history, among other operations in browser-style user interface.

These views were extremely useful since our Cluster nodes operating system was command line based, no GUI, and so, managing these services operations in an intuitive web page, instead of performing complex commands was much more accessible.

5.2.2 Cloudera CDH 5

As mentioned earlier to install Cloudera Impala, it is mandatory to deploy CDH distribution. We deployed the latest CDH 5. Like HDP's Ambari, to install CDH we used Cloudera Manager in order to automate the process of setting up and configuring of all essential components. Comparing this distribution with HDP, some of CDH's components are not as developed or updated as on Hortonworks.

For example, Hive on this distribution does not run the execution engine Tez, they didn't follow the Stinger initative, meaning, ORC File format isn't supported. Previously we decided to run Hive to HDP but only after we learned these differences regarding Hive's development, consequently, without doubt, Hive should not be tested in CDH. Cloudera seems to strongly focus on the development on Impala.

With Cloudera Manager we get centralized administration, like Ambari dashboard, both very detailed and useful with aggregated view statistics for all the services available (see Figure 21). The installation processes are similar, although when the deployment is done, there's always some post-installation configurations to do, unlike HDP, that from our experience usually all components ready to run right away.

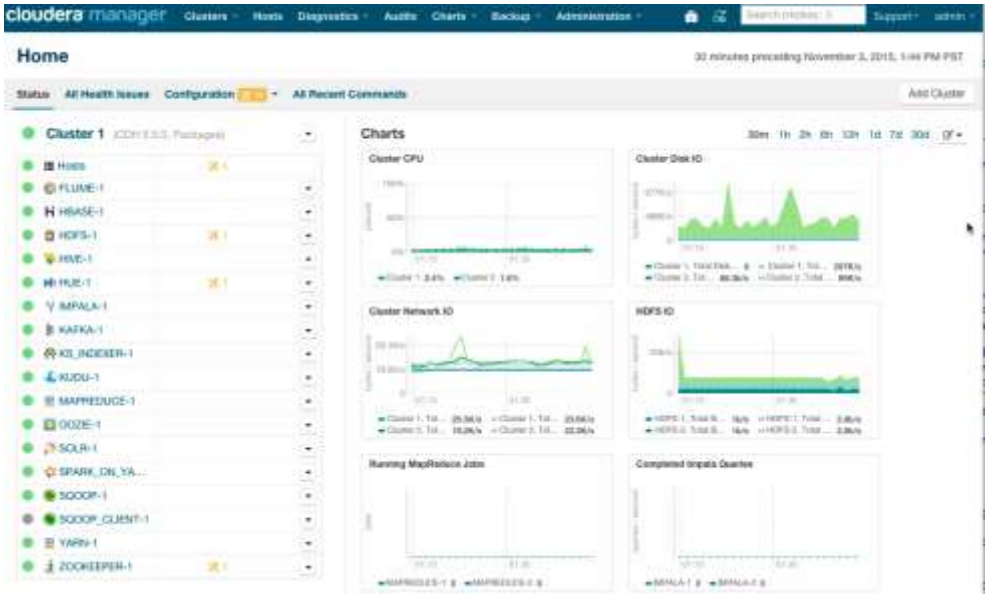


Figure 21. Cloudera Manager dashboard.

5.3 Benchmarking

To test the previously selected tools, we need a way to test their processing features on considerably large datasets to try to understand where each tool has advantages and weaknesses, performing a thorough experimental study. To measure performance, we conduct experiments to evaluate the effectiveness of the previously referenced tool using two Transaction Processing Performance Council (TPC) benchmarks.

Vendors use TPC benchmarks to illustrate performance competitiveness for their existing products, and to improve and monitor the performance of their products. For two decades, TPC benchmarks have been the gold standards benchmarks, quite popular when we need to compare databases and query performance.

In our work, we use TPC-H and TPC-DS benchmarks to measure query performance on the selected toosl using different scale factors (SF)[1] more specifically, we perform our performance evaluation using 10, 30, 50 and 100 GB datasets to observe scalability effects.

To measure the query performance of the chosen tools, we run the all the queries of TPC-H (see Appendix F) and a subset of TPC-DS queries (see Appendix G and Appendix H). We run the queries, 5 times each, and attempt to eliminate time variations by cleaning file system cache before running each query. The final query execution time is the average of the 5 runs.

5.3.1 An Overview of TPC-H Benchmark

The TPC Benchmark™H (TPC-H) is a decision support benchmark that provides a suite of business oriented queries, illustrating decision support systems that examine large volumes of data through the execution of queries with a high degree of complexity, using a

[1] Scale-Factor or SF defines generated databases or datasets sizes, corresponds to the raw data size.

variety of SQL operators, giving answers to common analytics scenarios and critical business questions [57].

Using the TPC-H benchmark requires the generation of the data schema and the population of the corresponding data, being afterwards possible to run a set of 22 business oriented *ad hoc* queries. Many database vendors have posted results on a vast range of hardware and at various scale factors up to 100TB. More details regarding the TPC-H benchmark can be found at the official website (http://www.tpc.org/tpch), where it is possible to find the DBGen tool, used for generating the several needed datasets for the benchmarks.

DBGen is an application deployed in C++ generating data files with a parameterized scale factor generating data in a very straightforward way. In a later stage, generated data is loaded to the TPC-H database schema, which consists of a 3rd Normal Form (3NF) schema with 8 tables (see Figure 22). Consider that LINEITEM and ORDERS is about 80% of the total dataset and that these tables would typically make up a fact table in a traditional, properly denormalised, data warehouse.

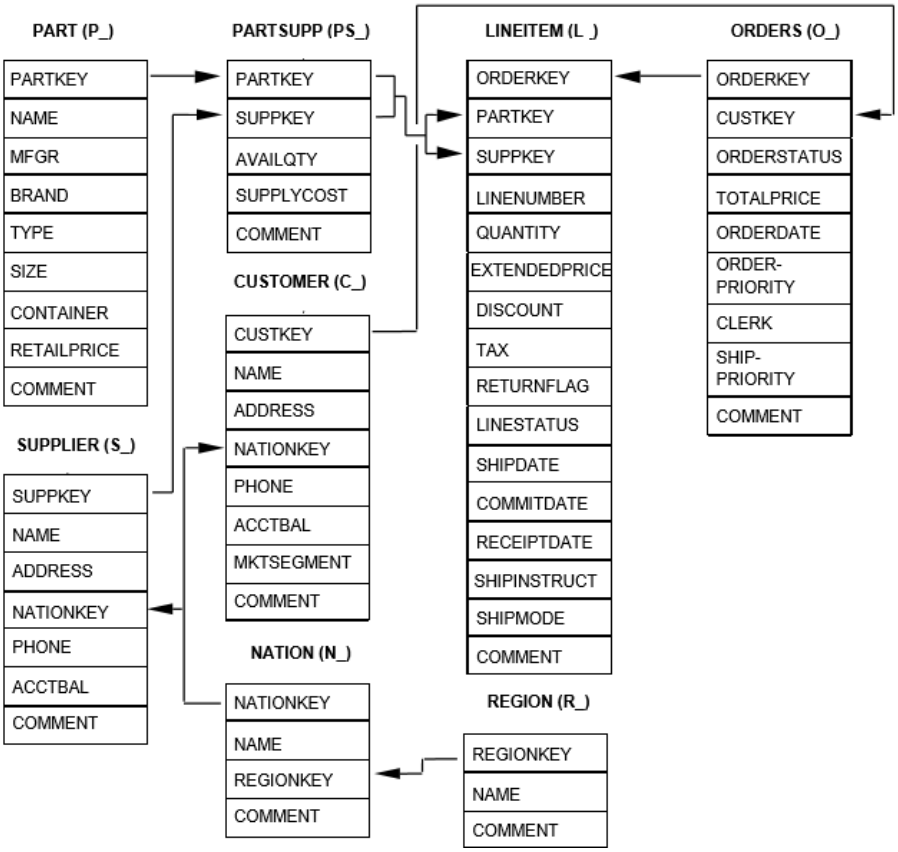


Figure 22. TPC-H schema model. Retrieved from [36].

5.3.2 An Overview of TPC-DS

The TPC Benchmark™DS (TPC-DS), is a decision support benchmark that models several generally applicable aspects of a decision support system that includes 99 queries operating on large volumes of data.

Although the underlying business model of TPC-DS is a retail product supplier, the database schema, and queries were designed to be representative of modern decision support systems, giving answers to real world questions.

The several queries cover various operational requirements and complexities (59 *ad hoc*, 41 reporting, 4 iterative OLAP and 23 data mining queries), this benchmark was certainly designed to test a wider range of features than TPC-H, originating high CPU and IO load during the processing. TPC-DS version 2 enables benchmarking on emerging technologies, such as Hadoop based systems [58].

From our knowledge, this benchmark is not used by vendors as much as TPC-H, maybe due to the high number of queries. In an early stage of our work we only planned to perform the TPC-H benchmark, but to complement the TPC-H evaluation, we decide to perform TPC-DS *ad hoc* and reporting queries. We chose to perform some of the TPC-DS *ad hoc* queries, since theTPC-H queries are also classified as *ad hoc*, consequently we can relate results of the two benchmarks. In adition, we also performed some of the reporting queries. We find relevant to observe how the chosen tools behave in the reporting portion of a decision support systems, answering well-known, pre-defined questions about the financial and operational health of a business. The official performed queries and respective descriptions can be found on Appendix G and Appendix H, more details about them can be found in [58].

The TPC-DS schema models the sales and sales returns process for an organization that employs three primary sales channels: store, catalogs, and the Internet. The schema is composed by several snowflakes with shared dimensions. Star and snowflake schemas are often found in data warehousing systems.

These schemas are represented by centralized fact tables connected to multiple dimensions, splitting up data into dimension tables, avoiding redundancy by moving commonly repeating groups of data into new tables. Each dimension has a single column surrogate key. The fact tables join with dimensions using each dimension table's surrogate key.

The tradeoff is an additional complexity in query joins. The TPC-DS schema models a data warehouse, consisting in 17 dimensions and 7 fact tables, a total of 24 tables with an average of 18 columns each, with vital business information such as customers, orders, and products.

For simplicity and space reasons, Figure 23 shows only an excerpt of the entire schema. It focuses on the store sales channel. For the entire schema and table details, please refer to [58]. More details regarding the TPC-DS benchmark can be found at the official website (http://www.tpc.org/tpch), where we can also find DBGen tool, used for generating data.

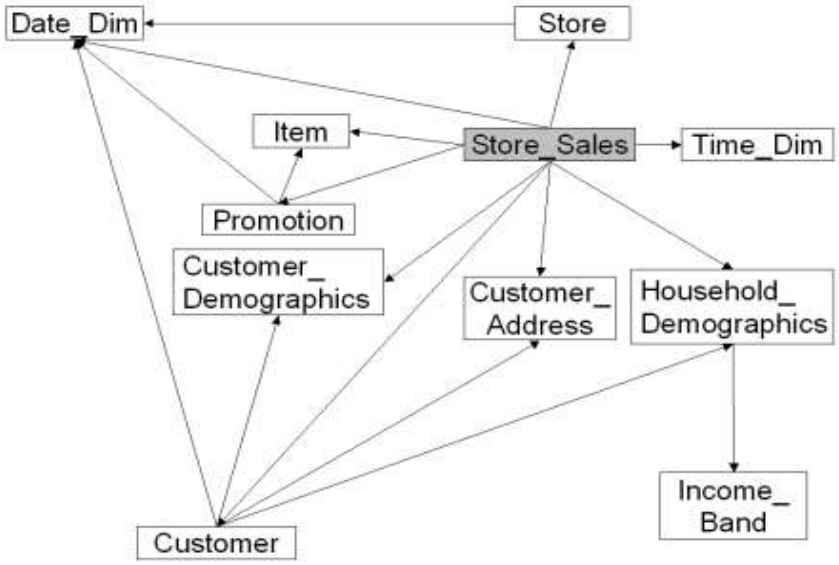


Figure 23. Store Sales schema. Retrieved from [58].

5.4 Storage Formats

In subChapter 4.7 we did an overall comparison of the analyzed tools on several aspects, among them was the supported file formats, an important factor since the data can be stored in numerous formats. Hadoop file formats such as Parquet and ORC, which can provide lightweight and fast access to compressed data with columnar layout, hence can significantly boost I/O performance.

As mentioned previously, the first tool approached was Hive, where we found that Stinger initiative improved tremendously its performance through the introduction of ORC file. All the other tools can support this file format as we have seen previously, except Cloudera Impala.

We used the ORC file format in Hive, and the Parquet file format in Impala which are the popular columnar formats that each system advertises [36]. Using ORC files brings advantages as efficient compression, leading to smaller disk reads. It also provides faster data readings, since it has a built-in index, min/max values, and other aggregates. These indexes also enable the skipping of irrelevant blocks of rows read by queries, also known as predicate pushdown.

Since we are working with TPC benchmarks, DBGen will generate data in native uncompressed text format, after this generation, we used Hive and HiveQL scripts to store and convert this data into ORC.

To do this conversion, we load the data into a suited Hive table (with proper structure/columns to store the data), and then create a ORC Table. With the ORC table created, we copy the data from the original text format table to the new ORC table, resulting in ORC format content. The following example shows a sample script to do the previsously described conversion:

```
CREATE TABLE test_details_txt( visit_id INT, store_id SMALLINT) STORED AS
TEXTFILE;
CREATE TABLE test_details_orc( visit_id INT, store_id SMALLINT) STORED AS ORC;

-- Load into Text table
LOAD DATA LOCAL INPATH '/home/user/test_details.txt' INTO TABLE test_details_txt;

-- Copy to ORC table
INSERT INTO TABLE test_details_orc SELECT * FROM test_details_txt;
```

Regarding parquet file format, it is an open source binary column-oriented data store of the Apache Hadoop ecosystem, like ORC, also providing efficient data compression, indexes and speeding up queries. According to Cloudera's benchmarking results, for purely I/O bound queries (queries that create bottlenecks in reading or writing to disk), they typically see performance gains in the range of 3-4x.

Parquet is compatible with most of the data processing frameworks in the Hadoop environment, including the tools that we previously analyzed, being eespecially good for queries which read particular columns from a wide table (with many columns), since only needed columns are read, I/O is minimized to store data as Parquet we used the same approach as the ORC conversion mentioned above, using the script like the previously referenced adapted create a Parquet Hive table (*CREATE TABLE…STORED AS PARQUET;).*

These file formats high focus on efficiency leads to some impressive compression ratios. The work [59] shows the sizes of the TPC-DS dataset at scale 500 in various file formats, including ORC and Parquet, where data in ORC gets 78% smaller than the original size. With Impala, 62% smaller than the original size. Since our cluster's capacity is limited, it is relevant to choose a file format that benefits higher compression rates. In Table 3 we show another demonstration of the files format compression capacities, presenting the sizes of data generated for the TPC-H for the SF 10, 30, 50 and 100GB in order to perform the benchmark. We can see the original uncompressed text format size by table and also the sizes resulting of the ORC and Parquet conversions, where we can see that ORC maintains the higher compression rate, with parquet maintaining good rates as well.

Table 3. TPC-H Hive table sizes by SF.

Tables/ SF size (in GB)	***Text (10)***	***ORC (10)***	***Parquet (10)***	***Text (30)***	***ORC (30)***	***Parquet (30)***	***Text (50)***	***ORC (50)***	***Parquet (50)***	***Text (100)***	***ORC (100)***	***Parquet (100)***
Part	229 MB	37.6 MB	122.2 MB	698.3 MB	112.9 MB	389.8 MB	1.15GB	190.3 MB	610MB	2.3 GB	376.2 MB	1.3 GB
Partsupp	2.277G B	293.5 MB	1.055 MB	3.4 GB	881.9 MB	3.3 GB	4.85GB	1.46 MB	5GB	11.4 GB	2.9 GB	10.9 GB
Supplier	13.4 MB	4.9 MB	14.2 MB	40.8 MB	14.7 MB	43.8 MB	65MB	24.5MB	70.1MB	136.3 MB	49 MB	146.1 MB
Customer	232.1 MB	77.2 MB	228 MB	702.7 MB	231.6 MB	701.8 MB	1.1GB	385MB	1.14GB	2.3 GB	771.8 MB	2.3 GB
Orders	1.652 GB	360.4 MB	1.095 GB	4.9 GB	1.1 GB	3.4 GB	8.4GB	1.8GB	5.4GB	16.6 GB	3.9 GB	13.6 GB

Tables/ SF size (in GB)	*Text (10)*	*ORC (10)*	*Parquet (10)*	*Text (30)*	*ORC (30)*	*Parquet (30)*	*Text (50)*	*ORC (50)*	*Parquet (50)*	*Text (100)*	*ORC (100)*	*Parquet (100)*
Lineitem	7.440 GB	1.6 GB	3.414 GB	22 GB	4.9 GB	11.6 GB	37GB	8GB	18.3GB	74.1 GB	16.7 GB	38.7 GB
Nation	2.1 MB	1.7 KB	3.2 KB	2.1 MB	1.7 KB	3.2 KB	2.1 MB	1.7 KB	3.2 KB	2.1 MB	1.7 KB	3.2 KB
Region	1.3 MB	1.0 KB	1.1 KB	1.3 MB	1.0 KB	1.1 KB	1.3 MB	1.0 KB	1.1 KB	1.3 MB	1.0 KB	1.1 KB

Analyzing the table above, we notice that some tables demand much more storage space, like Lineitem, Orders, these are tables with higher number of rows rows and columns. Table 4 presents the number of rows of each table by SF, so we can see the reason of size difference and complement table description.

Table 4. TPC-H Number of rows with different SF.

SF of TPC-H	*Customers*	*Lineitem*	*Nation*	*Region*	*Orders*	*Supplier*	*Part*	*Partsupp*
10GB	$1.5*10^6$	$60*10^6$	25	5	$15*10^6$	100 000	$2*10^6$	$8*10^6$
30GB	$4.5*10^6$	$180*10^6$	25	5	$45*10^6$	300 000	$6*10^6$	$24*10^6$
50GB	$7.5*10^6$	$300*10^6$	25	5	$75*10^6$	500 000	$10*10^6$	$40*10^6$
100GB	$15*10^6$	$600*10^6$	25	5	$150*10^6$	$1*10^6$	$20*10^6$	$80*10^6$

5.5 Loading and Accessing Data

As previously mentioned Hive was the first tool experimented, and since we wanted to load DBGen generated data, the first thing we had to do was transferring the data from the local disk to HDFS (achieved by running the command *hadoop fs -copyFromLocal /path/in/linux /hdfs/path*). Our generated data passes to HDFS and hence is accessible by Hive.

To access Hive, we can use *beeline* or *hive* command if we choose to access through CentOS command line. Since its friendlier, we used Ambari's Hive View (see Figure 24) where we created the schema using HiveQL.

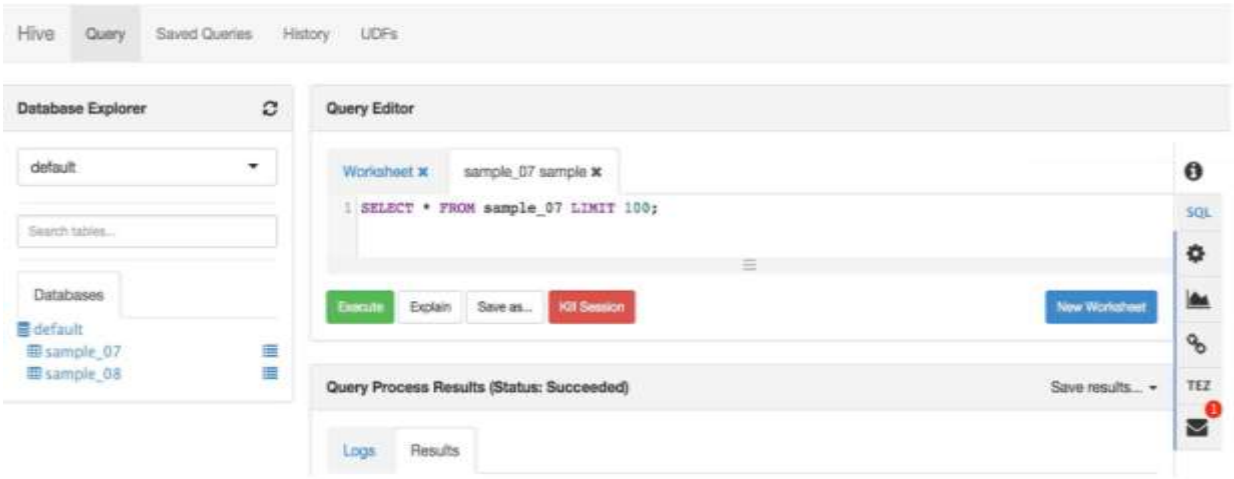


Figure 24. Hive View.

To create the Hive schema, we used a *CREATE DATABASE* statement, exactly like ANSI SQL. Then, we created the required tables (*CREATE EXTERNAL TABLE* statement, also similar to a classic SQL create table statement) that will store data in text format.

The next step was to load the text data to the created tables. Then, we create the ORC or Parquet tables using the statements similar to the ones presented in 5.4. With the ORC or Parquet table created, we copy data from the original text format table to the new table, resulting in ORC or parquet format content (also depicted in the scripts presented in 5.4).

We perform these operations for all the SF of TPC-H and TPC-DS, through the HiveQL scripts available on Appendix D and Appendix E, where the Parquet table creation part was only used on Impala. The creation table was performed on Hive as external tables, we could use a classic create table statement, but then data would only be accessible by Hive. Instead, we create external tables, so the data is accessible outside of Hive.

We followed this process since during our research we found that our tools can use Hive schemas through Hive Metastore and Hcatalog. We saw this fact as an opportunity to simplify the preparation and loading of benchmark data, concluding that it would be a time saving procedure to create Hive schemas and accessing them from other tools.

The analyzed tools have different methods to connect to Hive, this allow the SQL-on-Hadoop systems to connect and access Hive tables instead instead of raw files. For Drill, we used the already referenced Storage plugin interfaces, specifically, we configured hive storage plugin through Drill web console, that allows Drill to connect directly to Hive metastore, gaining access to data. Drill integration with Hive is only for metadata, Drill does not invoke the Hive execution engine for any requests.

To access Drill, we started the drill shell (sqlline command) from any cluster node containing a Drillbit and select Hive the schema, being able to query data contained in Hive, as shown:

```
0: jdbc:drill:> USE hive;
+------------+------------+
|     ok     |  summary   |
+------------+------------+
| true       | Default schema changed to 'hive' |
+------------+------------+
1 row selected (0.067 seconds)
```

Alternatively, we also used Drill Web UI to perfom queries, to perform the queries in a friendlier way. Similar to Drill, Presto provides the component Hive connector to access data stored on Hive. We used Hive connector that allows querying data stored in Hive. It does not use HiveQL or any part of Hive's execution environment. In order to query Hive data on Presto we used Presto CLI with the following command:

```
./presto --server presto_server_hostname:presto_server_port --catalog hive --
schema hive_schema
```

HAWQ uses PostgreSQL to process SQL statements, in order to enter PostgreSQL and use HAWQ'S features. We used the command psql connected to the node where HAWQ Master is installed, using nodes's hostname and HAWQ Master configured port:

```
psql  -h HAWQ_master_hostname -p HAWQ_master-port
```

To query Hive, we firstly opt by going through HCatalog integration with HAWQ and PXF, regardless of the underlying file storage format to reach Hive. This integration allows HAWQ to directly use table metadata stored in HCatalog, but querying approach has proven to be very slow and users reported that this approach suffer some performance degradation.

We decide creating native HAWQ tables from the existing HCatalog, replicating all the data stored in the Hive schema in HAWQ, from that point data is stored locally. Querying with the first option, PXF, HAWQ queried data with Hive engine, Hive would still be a bottleneck. To workaround this performance degradation we created a replication of the Hcatalog tables in HAWQ using the following SQL statement:

```
create table HAWQ_table as SELECT * FROM hcatalog.hive_schema.table_name
distributed randomly;
```

To access and query data with Spark, we used beeline command, same as Hive, but connected to the Spark component Spark Thrift Server. This component connects directly to Hive metatsore, from that point data is accessible by Spark. We can access Spark Thrift Server through beeline using the following command:

```
beeline -u jdbc:hive2://master_hostname:thrift_server_port/;httpPath=cliservice -n
spark
```

Regarding Impala, accessing Hive schema is also possible, since it can access directly to Hive metastore. In this case we aren't forced to use Hive to create the schema, we can do it directly in Impala, gaining better performance. Since we will be loading data from considerable large files, it will be an advantage.

In order to access and query data with Impala, we used Hue), a web-based interactive query editor included in the CDH components. Hue allows to visualize data, giving a complete view of complete schema and better table navigation (see Figure 25). Alternatively, we could use the command line to access Impala, using the command impala-shell).

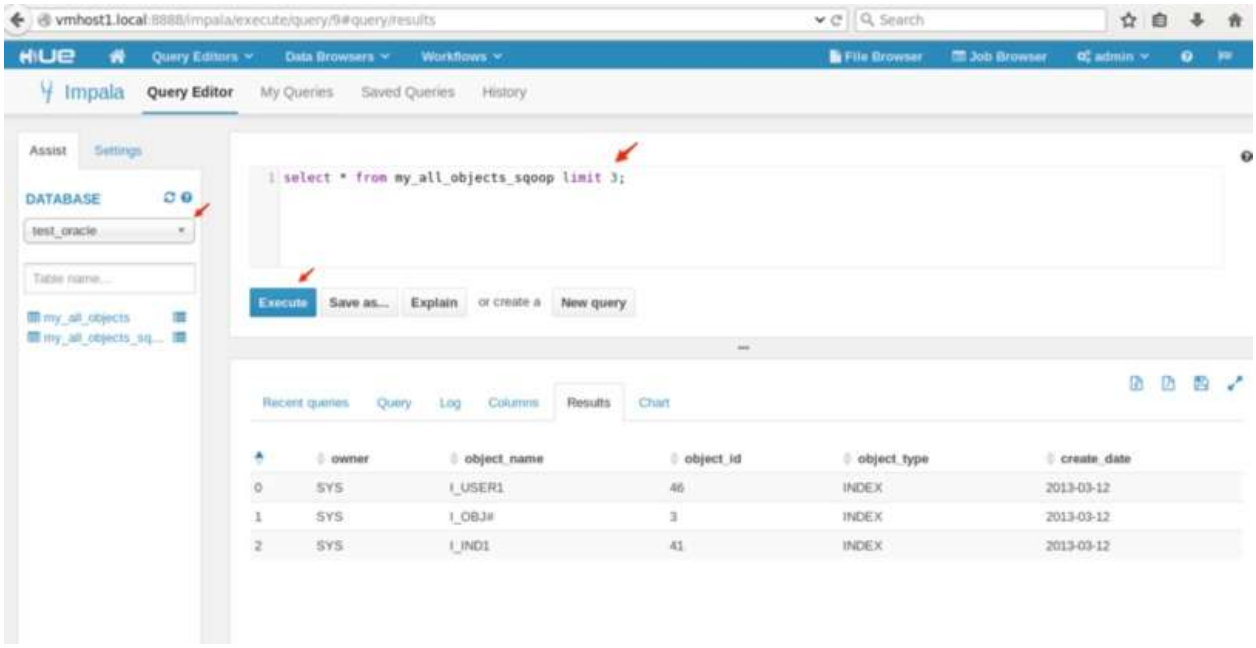


Figure 25. Hue Web UI.

In sum, all the described processed took some time and research, there was several ways to load and access data for each tool. The used approach allowed to save some time, we only had to build and load the schema on Hive. Since our set of tools can access Hive schemas, we saw an opportunity to simplify the process of preparing and loading data (without affecting each tool performance).

6 Experimental Evaluation

This chapter presents the performance evaluation of Drill, HAWQ, Hive, Impala, Presto and Spark using TPC-H and TPC-DS benchmarks. As mentioned before, we perform our performance evaluation using 10, 30, 50 and 100 GB datasets to observe scalability effects on performance. To measure the query performance of the chosen tools, we run the TPC-H queries and a subset of TPC-DS queries, 5 times each, cleaning the file system cache before running each query. The final query execution time is the average of the 5 runs. All these experiences were performed on the cluster described on chapter 1, that also details the used software and other decisions made.

6.1 TPC-H Benchmark

The complete TPC-H benchmark uses 22 queries on the previously presented table schema (subchapter 5.3.1). The official queries and respective descriptions can be found in Appendix F, and more details about them can be found in [57]. Some of these queries are not fully SQL compliant, especially those that need to use HiveQL, Hive and Cloudera Impala. In these cases, the queries had to be rewritten to be HiveQL compatible. These adapted queries can be found on the Hortonworks hive-testbench repository (https://github.com/hortonworks/hive-testbench/tree/hive14/sample-queries-tpch).

Some of these modifications from the official SQL compliant versions to the HiveQL versions include, for instance, the replacement of *SELECT TOP*, by *LIMIT* in the end of the query. Other modifications are due to the limited subquery support. As a workaround, the referred queries use views to replace the unsupported subqueries present in some queries like Q2, Q11, Q15 and Q22.

Before presenting the query execution times obtained from processing the several queries, Table 5 shows the tables that are needed in each query, in order to identify the ones that involve heavier JOIN operations. The results for TPC-H with scale factors (SF) of 10, 30, 50 and 100 GB are represented in Table 6, Table 7, Table 8 and Table 9. Overall, we can observe that queries with complex joins [2] and subqueries across several tables are the operations that are more demanding in terms of resources, requiring more processing time (e.g Q2, Q5, Q7, Q8, Q9, Q10, Q18 and Q21).

Additionaly, as and previously mentioned, Lineitem and Orders are the largest tables in the schema, representing 80% of the total size, therefore, we can expect that queries involving these two tables will have longer query execution times.

[2] A join combines rows from two tables or more by using the values of fields common to each of them.

Table 5. Table mapping by query.

	Customer	*Lineitem*	*Nation*	*Region*	*Orders*	*Supplier*	*Part*	*Partsupp*
Q1		X						
Q2			X	X		X	X	X
Q3	X	X			X			
Q4		X			X			
Q5	X	X	X	X	X	X		
Q6		X						
Q7	X	X	X		X	X		
Q8	X	X	X	X	X	X	X	
Q9		X	X		X		X	X
Q10	X	X			X			
Q11			X			X		X
Q 12		X			X			
Q13	X				X			
Q14		X					X	
Q15						X	X	X
Q16							X	X
Q 17		X						
Q18	X	X			X			
Q 19		X					X	
Q20								
Q21		X	X		X	X		
Q22	X							

Other costly operations include aggregation functions present in several queries (e.g Q1, Q10, Q13, Q18, Q20, Q21) and large IN clauses against a series of at most eight constant values (e.g Q12, Q16, Q19 and Q22). Also, when the queries use large amounts of data (which certainly happens when involving the biggest tables in the schema), operations like *ORDER BY* and *GROUP BY* will also have a large cost. As the scale factor grows grows, the used technological infrastructure starts to reach its limit, taking too long to process some queries or being, in some cases, unable to finish them, mainly for Presto, as can be seen in Table 8 and Table 9.

In the results tables, it is shown also the total query execution time to run all queries and the speedup. Speedup is determined as: slowest_tool_time/current_tool_time, are calculated.

Regarding the results for the 10GB SF, Table 6, we had no problem running all the queries, with HAWQ and Presto as the fastest tools with a speedup of 5.79 and 5.18, respectively, which means these tools are five times faster than Spark, which was the slowest one. Impala also presented good performance, being the third fastest tool running queries with query execution times ranging from 1.2 to 36 seconds and a speedup of 3.88. Although placed as third fastest tool considering the total time to run all the queries, Impala manages to be perform best on nine queries, surpassing the fastest tools HAWQ and Presto.

After HAWQ, Presto and Impala, Drill performed relatively well in the overall, except for Q19 and Q21 where the query execution times were similar to Spark. Hive performed better than Spark for most of the queries, except for queries Q7 and Q21.

In particular, Q21 is the one that requires more processing time, for all the workloads and all tools, since it involves a join across four tables (Supplier, Lineitem, Orders and

Nation), being the most demanding query of the benchmark (as the joins also include the two largest tables Lineitem and Orders).

Table 6. Query execution time for 10GB (in seconds).

	Drill	***HAWQ***	***Hive***	***Impala***	***Presto***	***Spark***
Q1	8.83	5.88	17.80	2.87	4.49	26.73
Q2	12.69	6.13	23.31	2.66	10.98	36.21
Q3	8.18	6.50	20.49	11.83	7.61	32.72
Q4	6.77	5.34	34.39	10.62	6.05	60.18
Q5	13.21	10.08	23.33	19.50	*10.6*	34.54
Q6	4.61	3.49	10.84	1.23	2.66	18.82
Q7	20.36	5.94	51.26	22.95	12.59	46.51
Q8	13.10	4.60	23.32	31.62	10.23	31.84
Q9	21.36	11.06	36.77	35.77	16.16	52.97
Q10	11.12	7.48	32.30	6.28	6.78	44.02
Q11	3.77	3.92	16.00	1.15	4.17	23.42
Q 12	8.53	1.15	14.82	2.21	4.03	25.14
Q13	6.42	4.52	26.39	10.23	5.97	68.68
Q14	6.49	3.92	13.86	1.78	2.87	23.39
Q15	12.97	5.52	19.47	4.25	3.16	30.51
Q16	12.70	6.46	17.78	1.51	5.88	30.28
Q17	9.20	15.62	16.22	1.68	8.41	23.85
Q18	32.77	14.83	42.21	12.18	14.40	86.33
Q19	31.01	2.21	20.51	3.16	4.41	33.85
Q20	11.52	7.93	19.89	8.98	3.87	26.79
Q21	96.39	18.99	128.08	32.24	25.18	106.92
Q22	7.18	4.29	25.21	7.95	3.66	38.94
Total	**359.18**	**155.86**	**634.25**	**232.65**	**174.16**	**902.64**
Speedup	***2.51***	***5.79***	***1.42***	***3.88***	***5.18***	***1***

Figure 26 shows the three fastest tools (HAWQ, Presto and Impala) and the obtained times for a representative query subset (Q5, Q7, Q9, Q17, Q18, Q21 and Q22).

This figure allows to see clearer the differences between the performances. Presto and HAWQ run with similar performance, surpassing Impala in all queries, except for Q17. In this query, we also see that Presto performs the query about two times faster than HAWQ. This query has some aggregation operations, but the heaviest workload resides on the associated subqueries. We see that Impala was also faster than the other two tools in Q18, with a two seconds difference. If we look at this query we see that the heavier workload resides on the aggregation calculations and joins, but since data fits in the available memory it could keep up with Presto and HAWQ.

For the remaining queries, we see that Impala needs much more time to execute the queries. In particular, performing queries Q8 and Q9 on Impala takes much more time due to a subquery that includes *JOIN* operations with six of the eight tables (including Lineitem and Orders).

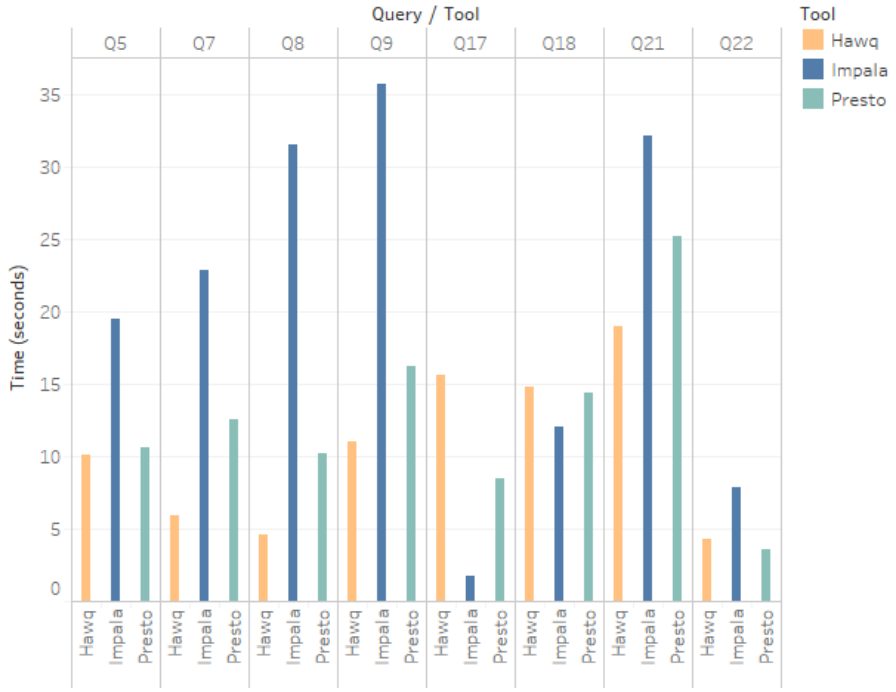


Figure 26. Sample query set for 10 GB between fastest tools (HAWQ, Presto, and Impala).

Table 7 presents the results of TPC-H queries for a scale of 30GB. Comparing with results using 10 GB (Table 6) query execution time is longer, but not linearly, meaning that for 30GB, query execution times was not three times slower than for 10GB. For HAWQ, Presto and Impala, the fastest tools in the previous SF, considering the total time required to run the 22 queries we see a decrease in the speedup of about 2.61, 1.7 and 1.95 towards the speedup observed in10 GB SF, meaning that the overall performance of the tools has less difference. For the current SF Presto managed to surpass HAWQ being now the fastest tool.

For Q1, which computes eight aggregates, a *COUNT*, four sums (*SUM*) and three averages (*AVG*), Drill was the slowest tool taking 2s more than Spark. Also in Q2, besides performing multiple joins, the query includes a *LIKE* string manipulation and a subquery that calculates the minimum cost part supplier for a certain part, in which Drill was once again the slowest tool. Although it performs the slowest times in those queries (Q1 and Q2), Drill was the third fastest tool, being able to surpass Impala and perform similarly to HAWQ and Presto in some queries like Q3, Q4, Q5, Q11, Q13.

In the previous SF=10 Impala was the third fastest tool, but for current SF=30 it presents longer query execution times in several cases surpassing 50 and 100 seconds, losing its edge on queries like Q3, Q5, Q7, Q9 and especially Q21. For these queries even Hive, one of the slowest tools is able to surpass Impala, except in Q9 where Hive took more 18 seconds to retrieve results.

We can see that in Q3 Impala was surpassed by Hive mostly due to the heavy performance of aggregation operations *ORDER BY* and *GROUP BY*, since this query *JOINS* three tables, including the biggest table Lineitem, consequently the intermediate results are huge, which incur large cost for the ORDER BY operation. Still Impala, shows very good performances in other queries, like Q11, Q12, Q14, Q15 and Q16.

Presto starts to be the quickest tool in the majority of queries, with HAWQ maintaining its good performance in most queries. In Presto, we were unable to run Q21, the remaining tools provided times ranging from 72.63 seconds (in HAWQ) to 213.8 seconds (in Impala).

As Presto uses RAM memory to process the data (in-memory processing, loading and reading data from RAM memory, instead pf the disk), and since JOINS present in this query process a considerable volume of data, as soon as runs out of memory, the processing stops.

Table 7. Query execution time for 30 GB (in seconds).

	Drill	***HAWQ***	***Hive***	***Impala***	***Presto***	***Spark***
Q1	30.93	15.2	25.78	13.21	13.10	28.82
Q2	58.12	13.32	36.86	11.36	13.43	38.47
Q3	20.97	24.47	29.72	52.50	16.72	36.75
Q4	18.96	21.01	85.66	51.49	12.31	125.66
Q5	28.79	26.67	45.00	68.46	27.59	49.29
Q6	7.70	8.83	10.24	5.57	3.47	22.61
Q7	42.67	13.66	76.40	78.68	34.18	82.52
Q8	17.19	12.21	32.56	12.11	26.68	41.86
Q9	27.68	25.22	182.58	164.2	37.75	126.68
Q10	27.72	23.26	62.17	17.72	14.36	63.17
Q11	4.22	8.88	23.68	6.88	9.14	52.15
Q12	14.86	20.11	29.65	5.55	11.28	27.59
Q13	12.97	7.92	54.60	30.84	9.83	62.82
Q14	12.95	19.14	21.02	5.85	6.51	33.83
Q15	25.8	13.46	24.18	3.93	6.68	38.66
Q16	14.84	10.05	25.45	2.77	7.04	30.70
Q 17	11.21	44.22	22.92	3.88	21.12	30.50
Q18	52.38	38.73	130.77	41.24	44.55	97.68
Q19	52.21	4.67	122.36	7.06	12.24	133.51
Q20	21.28	16.75	23.12	13.81	8.47	36.77
Q21	129.71	72.63	145.69	213.38	-	176.85
Q22	12.73	8.00	34.81	22.59	6.96	34.81
Total (w/Q21)	**516.18**	**375.78**	**1099.53**	**619.7**	**343.41**	**1194.85**
Speedup	***2.31***	***3.18***	***1.01***	***1.93***	***3.48***	***1***

Taking now a different set of queries (Q3, Q4, Q5, Q7, Q8, Q9, Q10, Q13, Q14, Q17, Q19 and q21), for the three tools that present an overall better performance for 30 GB (Presto, HAWQ, Drill), Figure 27 shows that although Presto is the tool in overall the tool that requires less overall time to run all the queries, still its surpassed in seven queries.

HAWQ, the fastest tool of the previous SF performs the best query execution times in the previously referenced seven queries (Q5, Q7, Q8, Q9, Q13, Q17 and Q19). In some of these queries, it performs with signigicant with significant time difference towards other tools, Q19 is the clearest case, where HAWQ performed the query under 5 seconds.

Also, HAWQ performs better in queries that have more complex JOIN operations and subqueries like Q7, Q8, Q9 and Q13 for this SF=30, also it was able to run the most complex Q21, being about 1.79 times faster than Drill.

Although Drill takes more time to run Q21 than the HAWQ, it is still an advantage to be able to run it, since our fastest tool Presto wasn't. On other queries, Drill was able to perform similar to Presto, in Q3 and Q5 and HAWQ, in Q3, Q4, Q5, and Q8. Drill is even able to surpass the fastest tool in Q17 with significant difference, on the other, along the also referenced Q21, it presents much slower execution time in Q19.

On the other hand, for queries Q7, Q10 and Q13, Drill presents the longer execution times, although with fewer time difference towards the other tools.

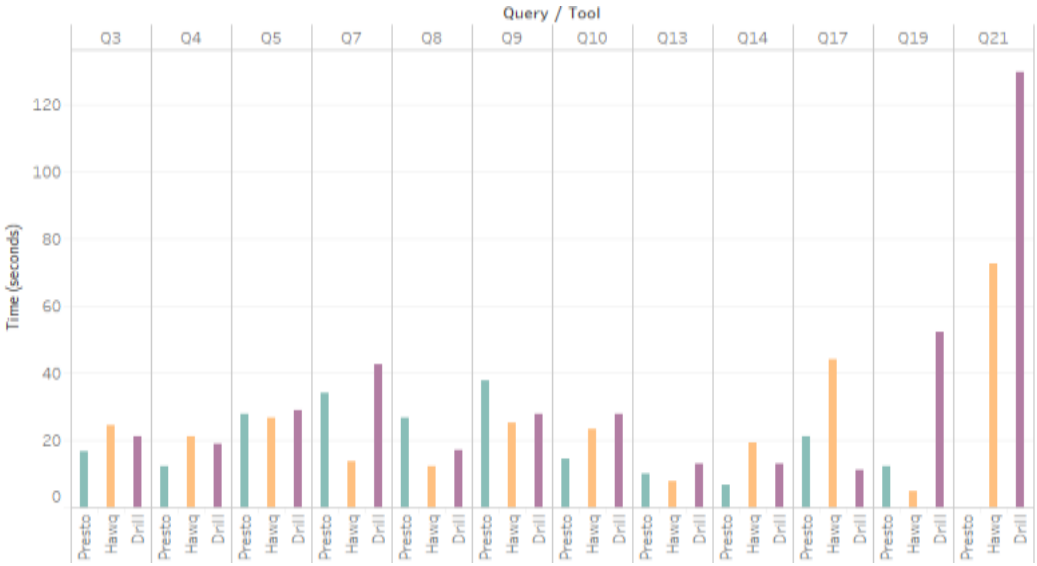


Figure 27. Sample query set for 30 GB between fastest tools.

In the 50 GB experiments, without considering Presto (since the three failed queries prevent the calculation of total execution time), observing the total time required to run all the queries we can see that Drill is the query tool that takes less time with some margin beating HAWQ and Impala.

The tools that we consider our lead runners (HAWQ and specially Impala) took a hit, being surpassed by Drill and presenting total query execution time similar to the slowest tools Hive and Spark. Still, we cannot ignore the fact that in some queries Drill, presents considerable delays.

In Q2, we see that Drill took about seven times more than HAWQ and Presto and 4.5 times more than Impala, Drill keeps struggling with the multiple joins performed by this query, being this case the only with execution above 100 seconds. Even Hive and Spark took less than half of the 113.16 seconds required by Drill.

Regarding Presto, although it couldn't run all the queries, it was the fastest tool on the major part of the queries performed, leading us to believe that if it could perform all the queries, it would be the tool that would take less time to run all the queries. Presto fails to query due to lack of memory, since it uses in-memory processing, at the moment that a Presto worker (one of the nodes), runs out of memory the query immediately fails.

Again, we hightlight that the execution query times of queries successfully ran by Presto surpass all other tools with significant difference, with times ranging from 5.45s to 54.35s.

We notice that HAWQ and specially Impala begin to suffer performance degradation, especially Impala. If we take for example Q9 and Q8, demanding *JOIN* operations continue to extend execution times. These operations are also the main repoonsible for query failures on Presto. For example, Q9 keeps failing on Presto, Impala performs the slowest time. From the usual fastest tools, only remained HAWQ that took nearly twice Drill's execution time.

Even though Drill takes less time to run all the queries, it wasn't able to run the complex Q21, like Presto. The remaining tools that were able to run this query took considerable amount of time to retrieve the results, where HAWQ took half the time Hive needed, followed by Impala.

Drill obtained its time advantage on queries like Q3, Q8, Q10, Q12, Q14 and Q17 but in the overall, although Presto fails to run all the queries, it still managed to be the fastest tool in 11 queries and Impala fastest in 8 queries.

HAWQ didn't get in the lead in any query, quite the opposite, being the slowest tool in 6 queries. Impala, the third fastest tool, managed to have the best performance in 8 queries, although it presents some exaggerated execution times, e.g Q3, Q4, Q7, Q8 and Q9 that presents execution times from more than 100 to 314 seconds.

Table 8. Query execution time for 50 GB (in seconds).

	Drill	***HAWQ***	***Hive***	***Impala***	***Presto***	***Spark***
Q1	43.23	67.91	34.34	22.59	17.82	37.32
Q2	113.16	15.99	49.34	25.54	15.11	48.29
Q3	36.01	63.44	40.65	109.48	23.93	49.08
Q4	24.97	60.63	96.41	132.21	20.84	63.54
Q5	51.46	61.75	56.47	107.33	33.47	53.61
Q6	11.43	51.02	12.62	8.75	5.45	24.94
Q7	79.89	59.22	105.49	146.15	54.74	115.05
Q8	29.22	62.57	56.53	286.85	46.39	52.92
Q9	42.56	79.62	194.22	314.97	-	191.05
Q10	36.16	73.32	85.04	29.61	20.19	91.02
Q11	4.52	14.21	24.41	8.15	10.48	44.78
Q 12	24.34	70.07	23.96	10.73	8.39	33.97
Q13	19.47	13.55	69.73	54.31	14.05	79.58
Q14	16.16	60.84	29.91	13.51	6.93	41.99
Q15	39.05	55.71	28.41	9.96	7.88	45.71
Q16	18.39	14.55	29.17	4.87	9.15	38.11
Q 17	17.01	158.72	30.05	8.69	27.81	38.33
Q18	93.66	111.62	214.47	84.79	-	142.56
Q 19	57.83	43.24	221.47	16.41	18.01	217.18
Q20	28.05	56.78	33.31	29.66	11.67	44.14
Q21	-	187.72	374.37	425.15	-	490.16
Q22	13.12	16.58	44.46	4.66	11.87	51.18
Total (w/Q21)	**799.69**	**1211.34**	**1480.46**	**1424.56**	**-**	**1504.35**
Speedup	**1.88**	**1.24**	**1.02**	**1.06**	**-**	**1**

We begin to notice deeply the effect of SF scability and higher data processed by queries containing *JOIN* across several tables and subqueries that extend execution times.

The query set presented in Figure 28, presents cases where Hive was faster than HAWQ like in Q3, Q5, Q6, Q8 and especially in Q17, where Hive was 5.26 times faster than HAWQ, performing similarly to Drill and Presto.

Regarding Drill, we can see that for this query set it beats all the tools in all cases, exept Q17, Q13 and Q6.

It is also impossible not to notice the exaggerated times performed by Impala in Q3, Q5 and sepecially in Q8 and Q9, where complex *JOIN*, *GROUP BY* and *ORDER BY* operations continue to delay query execution time.

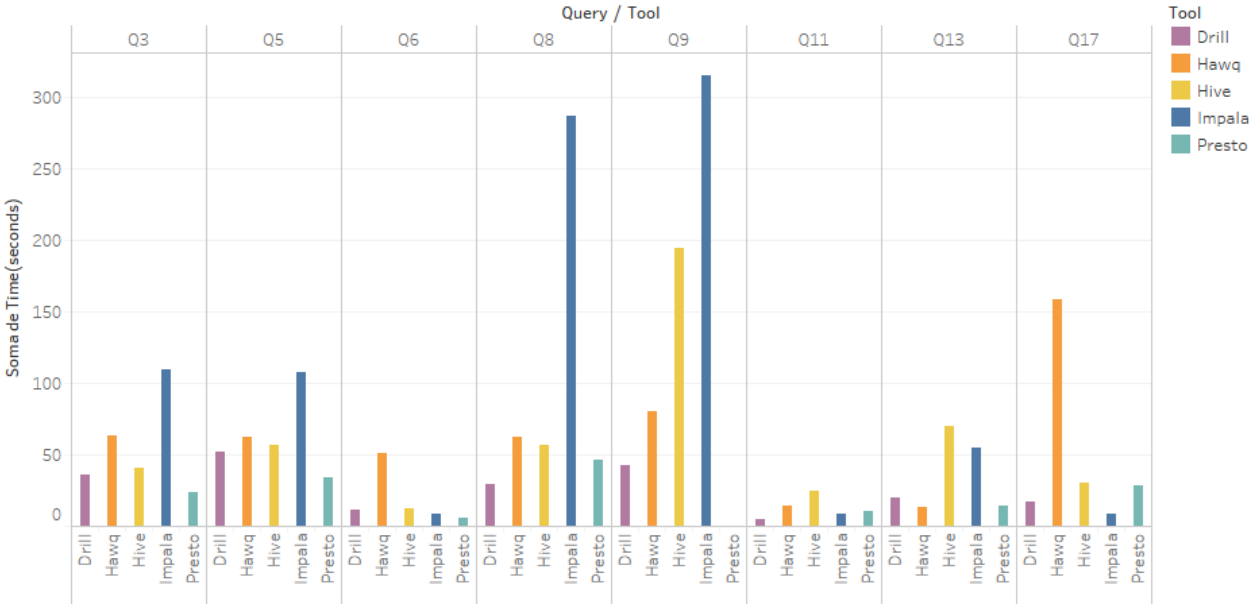


Figure 28. Sample query set for 50 GB.

In the experiments when we reach 100GB, the limit in terms of resources for our cluster is achieved. Figure 29 presents graphics that shows the resource utilization (CPU and RAM), retrieved from Ambari while we performed the benchmark for 100 GB, where we can see that these resources are under heavy load, even reaching 100% usage.

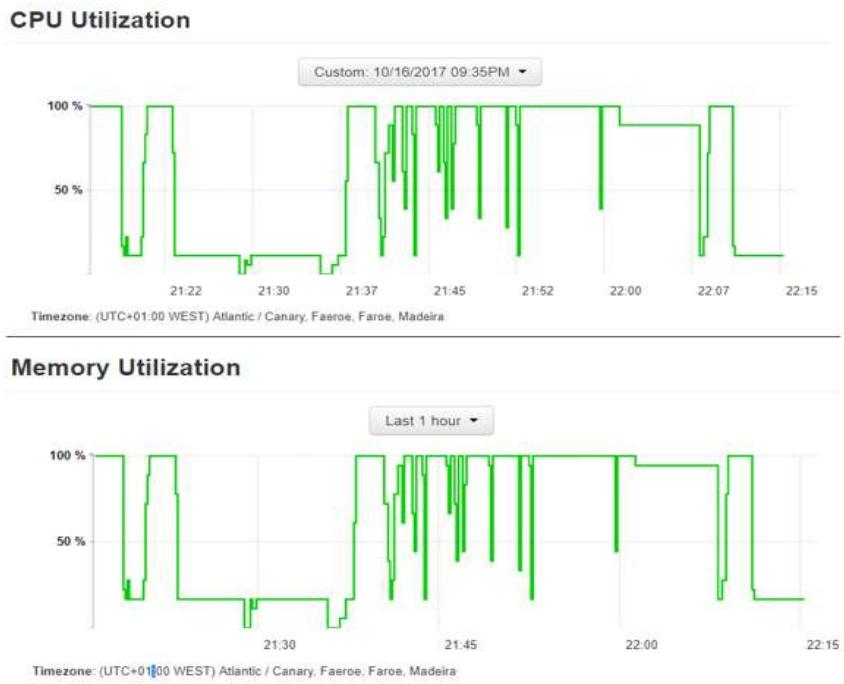


Figure 29. Cluster resource utilization (CPU and RAM).

We can see in Table 9, Drill was unable to process Q18 and Q21, while Presto was unable to run 7 queries (Q2, Q7, Q8, Q9, Q18 and 21) due to JOIN due to the voluminous JOIN operations. As previously stated, on queries that process a considerable volume of data, as soon as the tool gets out of memory, the processing stops. Since we are dealing with a considerable amount of data this happens more often.

Although Presto failed in many queries, it is worth mentioning that it was the fastest tool in almost all queries it could run (like Q1, Q3, Q4, Q6, Q10, Q12, Q13 and Q14). Exceptions can be found in Q11, where Presto was surpassed by Drill (nearly two times faster), Impala (about three times faster) and HAWQ (performed similarly being less than one second faster).

We observed similar situations on previous SF, leading us to believe that in a cluster with more resources (mainly RAM memory), Presto would be one of the top perfomers and a suitable solution for interactive querying.

Regarding Drill, like in the previous SF, kept similar performance to the fastest tools in some queries (e.g Q4, Q16, Q18, Q19). Also as in previous SF, Drill was able to surpass all other tools in some queries like in Q5, Q8 and Q9, meaning it's not a tool we should completely ignore although its surpassed largely by other tools in some queries. At this SF, we also notice that, for the first time Spark does not present the slowest processing times to run (slowest only in Q11 and Q16).

For HAWQ, increasing the workload keeps having a negative impact in its performance, having the worst times in several queries (Q1, Q6, Q12, Q14 and Q20). Nevertheless, was the fastest one in the most demandinf query (Q21).

Another aspect we must consider, besides query execution times, is the resilience of the tools. We notice that although Spark and Hive have not achieved the best results, they still managed to run all queries for this and previous SF, leading us to acknowledge their

robustness. In some queries, Hive present performances similar Impala and it even surpasses it in some cases like Q3, Q4, Q5, Q7, among others, although it was never able to surpass the times achieved by Presto.

HAWQ and Impala were also able to process all the queries, but we observe that they present a significant increase in the execution times. These two in-memory processing are very efficient if the processed data fits in the available memory. Otherwise, they apply a feature called "spill to disk", preventing queries that use memory-intensive operations from failing with out-of-memory errors (like we see in Presto), explaining the huge time differences that we see along the benchmark for Impala. In those cases, data is written in disk and the query does not crash, although it greatly impairs the performance. The work performed on [27] refered Impala does not handle large input sizes very well and now we know what why, although they didn't specified why.

Even though this feature granted robustness we could perform all the queries, increasing significantly the query execution time. The execution of Q9 and Q21 for the 100GB SF is a clear example of performance degradation caused by "spill to disk", where we can observe execution time on Impala above 500 seconds for Q8 and even 1000 seconds for Q9 and Q21. Although, for this queries HAWQ seems to manage these situations better, performing with great time advantage over Impala. Some other cases were already visible on the previous 30 and 40GB SF and keep degradation performance for the current SF e.g Q3, Q4, Q5, Q7, among other queries, that unlike SF 10 didn't provide almost instant query execution.

Table 9. Query execution time for 100 GB (in seconds).

	Drill	***HAWQ***	***Hive***	***Impala***	***Presto***	***Spark***
Q1	115.32	157.61	40.82	38.42	34.41	52.24
Q2	154.18	38.87	155.31	36.17	-	116.4
Q3	102.79	185.82	232.34	289.18	55.3	150.08
Q4	97.06	201.68	123.18	327.33	24.75	105.16
Q5	113.03	171.99	201.86	315.96	-	178.07
Q6	34.7	135.87	38.8	23.22	12.89	40.01
Q7	165.29	162.34	229.77	350.46	-	191.04
Q8	79.02	140.96	129.23	566.02	-	123.51
Q9	86.47	217.33	442.21	1070.66	-	420.5
Q10	98.55	150.81	259,13	60.52	46.04	143.69
Q11	16.35	28.65	31.63	10.64	29.78	44.09
Q 12	50.62	157.27	46.11	22.69	21.38	52.45
Q13	56.63	44.32	134.84	142.96	26.66	121.48
Q14	46.04	139.81	96.95	33.03	15.17	69.56
Q15	67.56	131.61	131.96	18.26	23.01	74.98
Q16	19.99	42.24	49.15	8.19	11.86	54.38
Q 17	46.4	396.57	69.91	8.82	86.06	61.67
Q18	-	280.87	373.75	226.01	-	262.55
Q 19	254.98	126.38	449.44	32.79	19.76	409.31
Q20	49.78	149.11	59.37	21.73	22.77	59.51
Q21	-	453.56	2870.82	1503.29	-	2869.76
Q22	17.2	41.84	96.19	62.82	14.37	76.44

Total	**-**	**3555.51**	**6262.77**	**5169.17**	**-**	**5676.88**
Speedup	**-**	***1.76***	***1***	***1.21***	**-**	***1.10***

Taking now some of the queries that all tools could process, Figure 30 shows how the query processing time is influenced by the type of query, as some tools are the fastest ones in some queries and can be the slowest ones in others. For this query set, we observe queries like Q1 or Q6, HAWQ seems to lose its edge being about 3x times slower than Hive and Spark, with Presto and Impala performing similarly to Hive.

When heavy aggregation operations are involved, the execution time is affected, but it is when we perform JOIN operations on huge volumes and subqueries that the tools really grasp to perform. In these cases, in-memory performance engines like HAWQ and Impala seems to lose its advantage if the data processed by the query doesn't fit in memory.

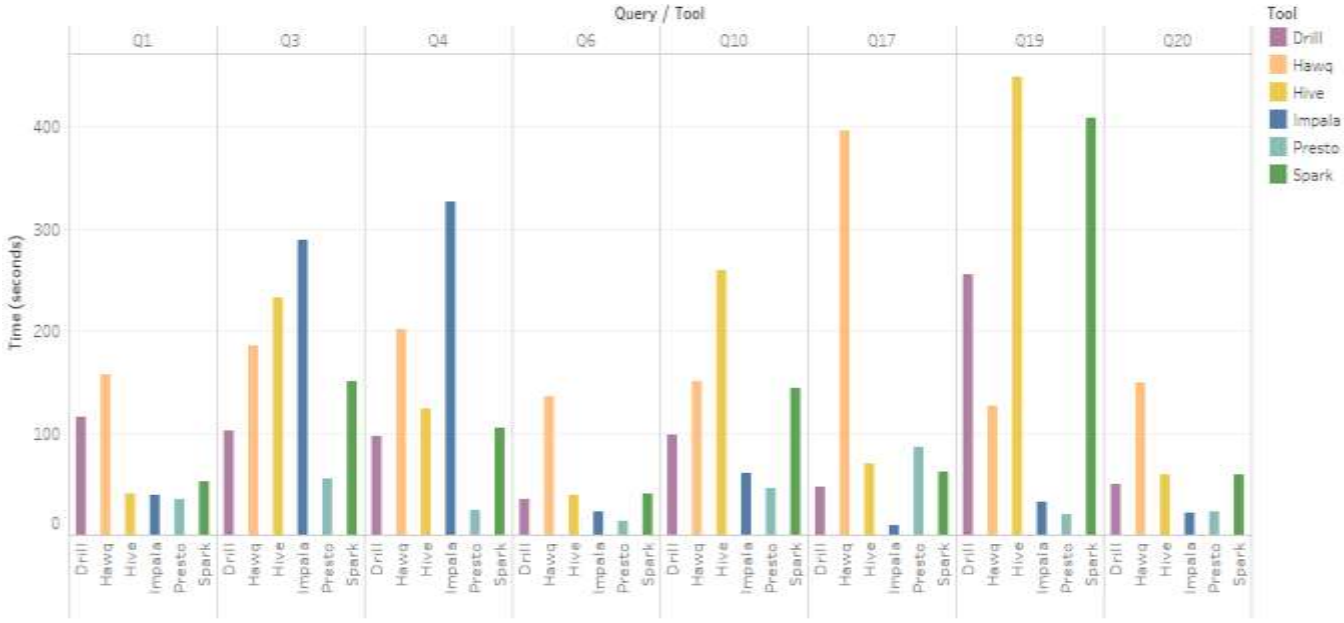


Figure 30. Sample query set for 100GB.

6.2 Main Findings Over TPC-H Results

In sum, we determined that in-memory processing tools like HAWQ, Impala and Presto can provide better performances. Many time during this evaluation we considered the total time to perform all the queries.

Nevertheless, there are cases that even though a certain tool isn't the fastest in performing the whole set of queries, it was still the fastest in single queries, and so, Figure 31 shows for how many queries the tools have achieved the fastest times. In this figure, we can see that although Presto in the 100GB SF couldn't perform some queries, its still the fastest tool on 10 queries, while Impalas was only the fastest on seven queries.

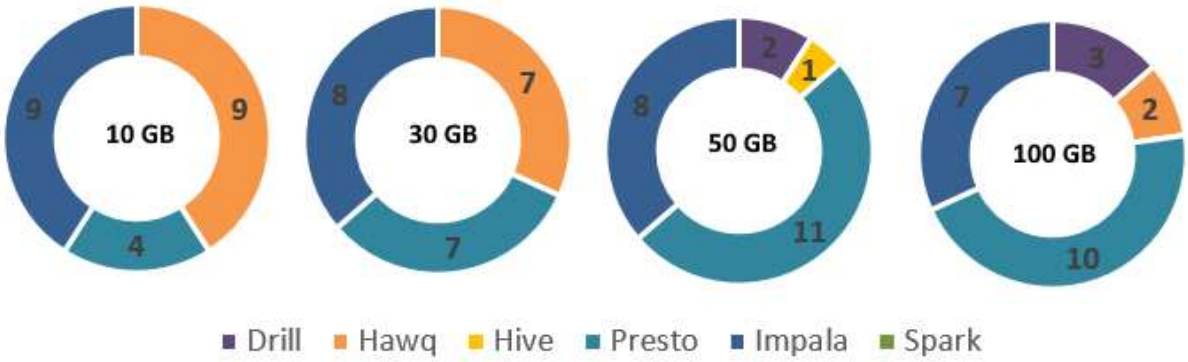


Figure 31. Number of Fastest Times by Tool

After comparing the results obtained we find that Hive are less suitable when querying in low data volume, due to the long coordination time of jobs. Altough, if we look at the different SF, as the data volume grows, Hive seem to catch up (less time difference between tools for the same queries) and even surpass the fastest tools on some queries, as we can see in Table 9 (where Hive surpassed mostly Impala and in some cases HAWQ for several queries).

When the volume of data starts to increase, the impact of the initial coordination time is less significant, consequently Hive's execution times get closer to the ones provided by the fastest tools. We observe the same tendency for Spark, although is always the slowest tool, similarly to Hive, it seems to achieve better results as the data scales.

Another factor that worth to point out, is the SQL compatibility. In

Table 2 we referenced the level of SQL compatibility of the several tools, where we can see that Drill, HAWQ and Presto have more extensive SQL compatibility, Hive and Impala use HiveQL, an SQL-style query language that isn't fully SQL compliant, consequently it does not support all SQL features. This was a problem that emerged when we were running the benchmark queries, mostly in cases that involved subqueries.

To surpass this, we had to use adapted queries that include subqueries in the WHERE clause, since not all types of subqueries on *FROM*, *WHERE* and *HAVING* clauses are supported. In these cases, these subqueries where replaced by SQL Views to store the data and then use that view in the original query instead of the subquery (this change was made for Q2, Q11, Q15 and Q22).

Like previously referenced, Spark was the slowest tool in all the SF's, which we did not expect since like the fastest tools it uses in-memory processing. We did some research in order to find if were somehow imparing Spark performance by using our approach, or if we had less correct configurations. We conclude that our approach was valid and should not influence spark performance. Still, we found that the users often use the dataframes (SparkSQL feature described in 4.6).

The use of this feature might aid improving Spark performance, but in order to do this we would have to build a scala script (or any programming language supported by spark), create the necessary variables to access the data schema, create the dataframe and load with

the data, finally use that dataframe to apply an sql query (*val results = hiveContext.sql("SELECT * FROM hive_table"*). As we notice this is a much more complex process and it implies much more that running a simple SQL query (like we did in all other tools) and so, like stated in [60] Spark general-purpose is not to be SQL layer for interactive/exploratory analysis, instead its more suited to build scripts in structured languages like Scala with SQL constructs.

In our opinion, supported by the theoretical investigation and the performance evaluation using TPC-H benchmark, in terms of query execution time, the best tools are Presto, HAWQ, and Impala being the use in-memory processing a huge advantage in query execution.

These are the tools that show the best performance in the execution of TPC-H queries and consequently best speedup.

Although Presto runs out of memory for some queries, mainly in the 100 GB SF, in terms of performance it is the most suitable tool for interactive querying, in-memory processing reduces query execution time significantly with the advantage of being fully ANSI SQL compliant.

According to the experimental evaluation and the results of query execution time for the different SF, we predict that if we our cluster had more resources, especially more RAM memory, Presto would outperform all the others, since Presto was the best performer, or one of the best performers in all queries that it possible to run.

Impala and HAWQ also use in-memory processing, we observed that these tools deliver very good performances, especially HAWQ. Both of these tools have activated by default the already referenced and described “spill to disk”, when data processed by queries doesn’t fit in memory.

This feature assured that all queries ran successfully, even if data does not fit in memory, and ensures query completeness, rather than failing with an out-of-memory error, the tradeoff is decreased performance due to the extra disk *I/O* to write the temporary data and read it back.

Still, along the benchmark we observed that HAWQ seemed to perform better than Impala in the use of “spill to disk” (although, the feature always delayed query execution time, HAWQ was usually faster). SQL clauses that more require memory allocation that could activate the spilling mechanism include *GROUP BY* clause for columns with millions or billions of distinct values, *JOIN* operations over large tables and sorting operations performed by ORDER BY clause.

Overall, we believe that Hive is still a great tool. Hive is seen like a standard, it was the first to support SQL-on-Hadoop, in our opinion is the most mature of the tools.

In the performed TPC-H benchmark it was always able to run all the queries, and so we must aknowledge its robustness.

Still, observing the results retrieved, Hive is more suited for batch processing, not recommended to process low volumes of data or scenarios where quick query execution times are mandatory. It can be applicable when low latency/multiuser support is not a requirement, such as for batch processing/*ETL*. Spark’s performance was almost in all times

the slowest tool, and probably more suitable when the objective is not just to query data, but to perform algorithmic analysis, statistics and Machine Learning. Drill provided relatively good performance, in most cases it couldn't keep up with the fastest in-memory processing tools. When the amount of data processed by the queries start to overwhelm the amount of memory, Drill managed to surpass some of the fastest in-memory tools. Still in our opinion tools like Impala, Presto and HAWQ are the tools more suited to perform ad hoc queries and interactive analysis.

6.3 TPC-DS Benchmark

The TPC-DS benchmark includes 99 queries. Unlike TPC-H queries there was no need of major query rewritten to be HiveQL compatible, only the replacements of TOP statements by HiveQL LIMIT. To address the enormous range of query types and user behaviors that can be found in a decision support system, TPC-DS uses a generalized query model. This model defines four broad classes of queries that characterize most decision support queries:

- Reporting queries
- Ad hoc or interactive queries
- Iterative OLAP queries
- Data mining queries

Perform the whole TPC-DS benchmark would require a significant amount of time, and so we are only performing a subset of interactive queries and reporting queries to complement the previously performed *ad hoc* decision support TPC-H benchmark.

The catalog sales channel is majorly dedicated for the reporting part, while the store and web channels are dedicated for the *ad hoc* part. Regarding the schema, the heavier tables to process include those who store sales information, Catalog_Sales, Store Sales, Web Sales and Inventory tables. For more specific information about the stored information and the tables structure please refer to [58].

In this schema fact tables are heaviest to process, where Store_sales and Inventory tables are the ones that require more space, meaning they store more rows, so we assume that queries involving one of these tables will have higher query execution times.

6.3.1 *Interactive* queries

Interactive or *ad hoc* queries capture the dynamic nature of a decision support system, queries are constructed to answer immediate and specific business questions. TPC-DS interactive queries join over single fact tables, which may involve advanced SQL features such as windowing functions or rollups (useful in generating queries that contain subtotals and totals) [59].

When we performed the TPC-H we observed that query performance is highly affected by heavy JOIN operations, arithmetical and aggregation functions. In the case of TPC-DS,

Table 10 presents the several tables and the queries in which they are used. The ANSI SQL interactive queries can be found in Appendix G.

Table 10. Table mapping by interactive query.

	Q 3	***Q 7***	***Q 12***	***Q 15***	***Q 18***	***Q 19***	***Q 26***	***Q 27***	***Q 42***	***Q 43***	***Q 52***	***Q 55***	***Q 82***	***Q 84***	***Q 91***	***Q 96***
Item	***X***	***X***	***X***		***X***	***X***	***X***	***X***	***X***		***X***	***X***	***X***			
Catalog_Page																
Call_Center															***X***	
Catalog_Returns (fact)															***X***	
Catalog_Sales (fact)				***X***	***X***		***X***									
Customer				***X***	***X***	***X***								***X***	***X***	
Customer_Address				***X***	***X***	***X***								***X***	***X***	
Customer_ Demographics		***X***			***X***		***X***	***X***						***X***	***X***	
Date_Dim	***X***	***X***	***X***	***X***	***X***	***X***	***X***	***X***	***X***	***X***	***X***	***X***	***X***		***X***	
Household_ Demographics														***X***	***X***	***X***
Income_Band														***X***		
Inventory (fact)													***X***			
Promotion		***X***					***X***									
Reason																
Ship_Mode																
Store						***X***		***X***		***X***				***X***		
Store_Returns (fact)																***X***
Store_Sales (fact)	***X***	***X***				***X***		***X***	***X***	***X***	***X***	***X***	***X***			***X***
Time_Dim																
Warehouse																***X***
Web_Page																
Web_Returns (fact)																
Web_Sales (fact)			**X**													
Web_Site																

Regarding the 10GB dataset, the obtained results are presented in Table 11 where we can observe reasonable execution times. Even though, like in the TPC-H benchmark, Spark keeps being the slowest tool and being surpassed by Hive in every query. The fastest tools for this SF were Impala and HAWQ, where Impala surpasses HAWQ in all queries except for Q43. Although Presto was always behind Impala and HAWQ with significant differences, it was able to perform all the queries with fast execution times, since most of the queries ran under 10 seconds.

We can see that these tools achieved short execution times, especially Impala reaching times below one second in some queries like Q12, Q42, Q55 and Q96, with an overall speedup of 11.66 towards the slowest tool Spark. The fastest tool Impala, has an above average execution time (comparing with other queries) when executing Q82, a fact also verified for the second fastest tool HAWQ.

It is also the longest query runned on Hive and Spark, taking nearly three times more than Drill, HAWQ and Presto. These greater execution times can be explained since the query performs *JOIN* operations between the two biggest fact tables, Inventory and Store_sales (being store the channel where more sales are performed) and the dimensions item and date_dim.

These joins result in the processing of eight milion rows that have several filters in the *WHERE* clause, reducing to five milion. Afterwards, *GROUP BY* groups the rows resulting in a 2.5 milion rows, from which the user will only see 100 rows since query is limited to show the top 100 rows.

In the previous TPC-H, we noticed some latency in Drill when queries evolve aggregation functions like *SUM*, *AVG*, and *COUNT*. Consequently, it is relevant to see if this tendency maintains. On the current scenario, similar to what we observed on the same SF on TPC-H, Drill couldn't keep up with the fasters tools. Consequently, when we look at the time required to perform all the queries, Drill was almost six times slower than Impala and almost four times slower than HAWQ.

For Drill, Q3 is the query that needs more time to run, perfoming similarly to Hive, one of the slowest tools for this SF (Drill performed the query about two seconds faster). Analyzing HAWQ and Presto execution times, for these tools, this query also requires above average execution time comparing with times performed on other queries. On the other hand, Impala, the fastest tool for this SF retrieved results from this query very fast, perfoming the query in near one second.

Analyzing Q3, it performs a join between the biggest fact table Store_sales and two dimensions, date_dim and item (processing a total of 7.9 milion rows). Each of these dimensions is filtered by month, date of sale and ID of the item manufacturer (for more details please refer to Appendix G). This query also in perfoms a *SUM* function over a fact table column to calculate the the total sales price per item brand of a specific manufacturer for all sales in a specific month of the year.

The same situation in Q7, where Drill executes the query with a performance similar to Hive (less than 1 second faster than Hive), being the query that needs more time to run after Q3. In Q7, four AVG functions are performed over four Store_Sales columns, joining data from more four dimension tables (customer_demographics, date_dim, item and promotion) performing grouping (*GROUP BY*) and ordering functions (*ORDER BY*) over the returned rows from Items table, to retrieve the averages for promotional items sold in stores, where the promotion is not offered by mail or special event.

Once again, this proximity of Drill's performance with Hive in queries that include aggregate functions can reveal stress on performing queries that include this kind of operations. Another reason is the high quantity of processed rows.

Once again taking Q7 as example, this query joins a total of two milion rows that include *WHERE* clause filters (reducing rowset to 164716), lastly the aggregate and grouping functions will give a result of 85000 rows.

Table 11. Interactive Query Execution Time for 10 GB (in seconds).

	Drill	***HAWQ***	***Hive***	***Impala***	***Presto***	***Spark***
Q3	24.06	7.83	26.25	1.05	13.82	34.51
Q7	18.67	4.36	19.05	2.44	10.55	28.44
Q12	11.17	3.49	14.37	0.86	6.78	20.28
Q15	13.17	7.08	16.98	1.65	9.18	22.73
Q18	20.22	5.84	29.51	3.64	10.57	37.65
Q19	15.81	4.27	16.45	2.22	10.25	25.97
Q26	12.87	3.22	19.34	2.17	8.27	26.36
Q27	13.71	3.12	24.64	2.29	9.49	28.32
Q42	8.28	1.48	6.83	0.84	8.07	17.91
Q43	10.19	2.82	23.28	4.47	9.12	30.58
Q52	8.39	1.87	8.51	1.06	7.52	17.56
Q55	8.83	1.63	10.39	0.71	7.63	17.38
Q82	11.73	10.25	33.42	8.47	10.05	43.64
Q84	8.82	4.26	24.13	2.78	6.27	32.03
Q91	12.36	5.03	15.42	1.96	7.94	22.17
Q96	8.73	1.76	20.32	0.55	7.55	27.58
Total	**207.01**	**51.74**	**308.89**	**37.16**	**143.61**	**433.11**
Speedup	**2.09**	**8.37**	**1.40**	**11.66**	**3.02**	**1**

Taking a set of queries, for the three tools that present an overall better performance for 10 GB (Presto, HAWQ and Impala), Figure 32 shows that Presto is the tool that needs more time to run all the presented queries, except Q82 where Presto and HAWQ performed nearly the same time. In this query, Impala performed best, being less than a second faster.

Query 82 joins the dimensions Date_dim and Item with the two largest tables, Store_sales and Inventory causing I/O intensive scans of vast amount of data, consequently it is the most demanding query in this set. Impala performs all the queries with significant advantage towards the other tools although, when running Q82, the query execution time is above average, but it was able to perform the query in 8.47 seconds, making it the faster tool, still it was the tool that run the query in less time. Performing Q3, Impala is more seven times faster than HAWQ and nearly fourteen faster than Presto.

For Q43, HAWQ has the best performance (two times faster than Impala), unlike all others where Impala performs best. This query joins the biggest fact table store_sales and dimensions date_dim and store, processing a total of 7.9 milion rows, then after filtering this rowset, performs seven SUM functions over Store_sales fact price column, including *GROUP BY* and *ORDER BY* function, retrieving 1.1 milion rows.

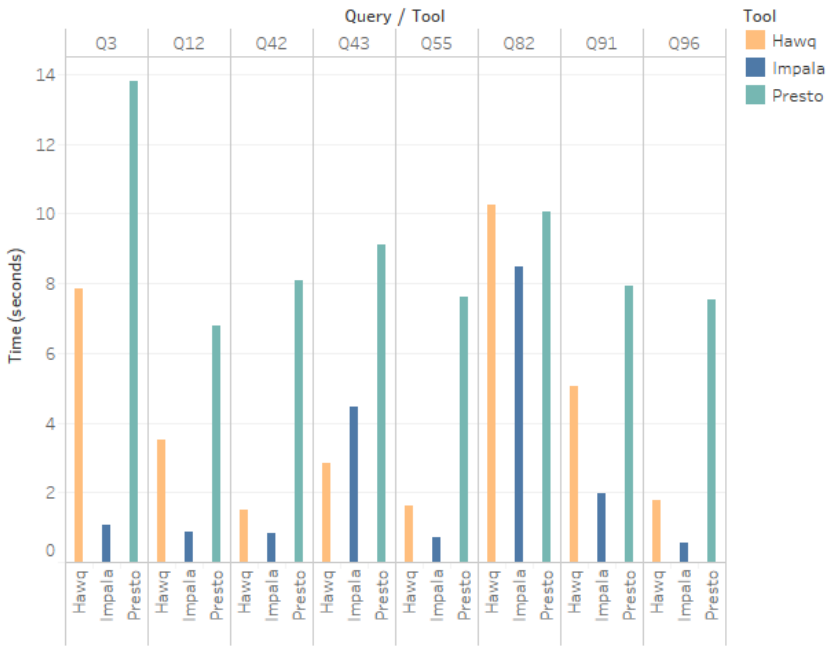


Figure 32. Interactive Sample Query Set for 10 GB between the fastest tools.

For the 30GB SF, one more time we observe that the execution times don't increase linearly, since we don't observe 3x increase of execution times of 10GB. Similar to the 10GB sample, Table 12 shows that HAWQ and Impala manage to be the fastest tools. HAWQ was the quickest performing all the queries, surpassing Impala, the top performer of the previous SF. So, if we consider the total time to execute all queries, the fastest tools for this SF are HAWQ, Impala and Presto.

With the growth of SF, HAWQ surpassed the 10GB fastest tool Impala, managing to be fastest in Q7, Q18, Q19, Q27, Q43 (in this case nearly 50% faster) and Q84 for this SF. Hive kept surpassing Spark in every query having a 1.40 speedup regarding Spark. Presto was again the third quickest tool, managing to have good performance in all queries, being surpass by Impala in every query except the demanding Q82.

Like the previous SF, Q3 is one of the queries with longest execution time (only surpassed by Q82), independently from the tool, it retrieves results in above average times, comparing with other queries times. The difference is more noticeable on Presto and Drill and especially on HAWQ.

Q7 also keeps running on above average query execution time on Presto and especially on Drill, this query now processes fourteen milion rows, resulting from the several previously described joins, then applying filters to apply aggregates, ordering and grouping functions (ORDER BY and GROUP BY), due to the high number of processed rows, these functions represent a performance bottleneck.

Table 12. Interactive query execution time for 30 GB (in seconds).

	Drill	***HAWQ***	***Hive***	***Impala***	***Presto***	***Spark***
Q3	27.13	19.39	31.27	8.95	17.84	36.52
Q7	21.06	6.86	23.39	7.21	17.78	32.55
Q12	13.34	6.61	16.04	3.17	11.74	22.81

Q15	17.47	14.15	17.83	6.05	12.04	25.09
Q18	25.27	7.33	39.56	7.86	13.52	43.83
Q19	23.33	5.61	18.49	6.39	14.92	27.09
Q26	16.26	3.44	23.32	2.59	11.75	31.16
Q27	20.42	2.29	26.35	3.41	15.04	30.26
Q42	11.86	2.64	10.19	2.59	11.37	20.89
Q43	14.44	5.32	26.72	10.15	15.36	33.98
Q52	10.64	2.57	12.78	2.33	11.22	22.18
Q55	10.98	2.62	14.44	2.58	11.15	21.28
Q82	14.96	24.09	35.57	19.79	10.77	48.96
Q84	10.73	5.45	27.54	6.54	7.47	34.78
Q91	17.45	6.28	17.88	3.12	12.44	23.42
Q96	10.78	2.52	24.44	1.82	9.94	29.98
Total	**267.02**	**83.63**	**365.81**	**94.55**	**204.35**	**484.78**
Speedup	**1.82**	**5.79**	**1.33**	**5.13**	**2.37**	**1**

Picking a new set of queries that all tools could process, Figure 33 shows the times performed by Drill, HAWQ, Hive, Impala and Presto. Spark was left out of this figure since it maintained the status of slowest tool.

On the other hand, differences between Hive query execution times and remaining tools start to be less significant (as we can see in Q7, Hive only needs about two seconds more than Drill to execute). Although it still needs to perform much better to keep up with the best tools. It maintained as the second slowest tool after Spark and the slowest tool for the set represented in the figure below.

In the previous analyzed 10 GB SF, we identified Q82 as one of the more complex query in terms of processing, being the fastest tools to run it Impala and Presto (in this order), with very few difference. For the current SF Presto was the fastest tool.

With the growth of data processed the time difference became more significant, Presto perfomed best, beating Impala in nearly 9 seconds. This significant delay allowed Drill to surpass Impala, performing the query in about 14 seconds seconds. Still in Q82, Drill managed to be the second fastest tool, with a difference of about fout seconds towards Presto.

Regarding Q18, like the times observed on 10GB SF, this query has the longest running time time on Hive and Drill and the second slowest query on Drill, while the remaining tools provided fast execution times

This query joins data from fact table Catalog_sales and four dimention tables (processing 1 milion rows), performing seven AVG functions over Catalog_Sales columns, where this tool seems to take more time to perform. In Q43, Q42 and Q55, although with very few difference, Drill managed to surpass Presto. In Q42 and Q55, Presto even presents almost the same performance as one of the slowest tools Hive. These performance proximities should keep us alert to these tools performance in the next SF scale.

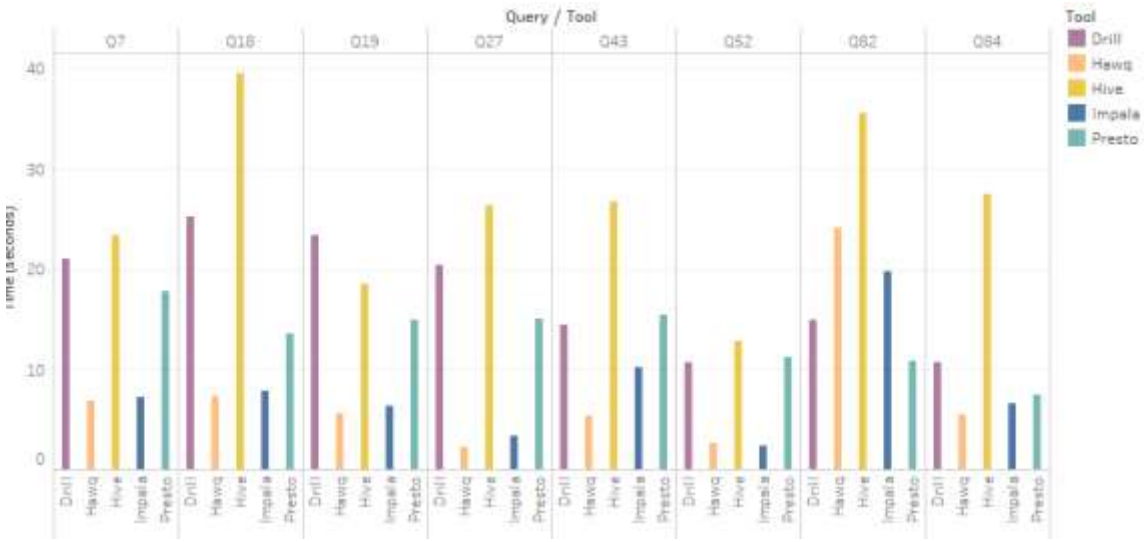


Figure 33. Interactive Sample Query Set for 30 GB

When we run the benchmark on 50GB SF, HAWQ keeps leading the execution times followed by Cloudera Impala and Presto as presented in Table 8. Drill kept following Presto closely, performing all queries in 341.07 seconds against the 299.21 from Presto.

Still, Drill is able to perform similarly to the fastest tool HAWQ on queries Q3, Q12, Q15, Q18, Q27, among others. Although its out of the top three fastest tools we noticed that, for the demaning Q82, it's the second fastest tool, being only surpassed by Presto.

As usual, Spark kept as the slowest tool, surpassed by Hive in every query. Impala keeps being the second fastest tool, still it manages to be the fastest tool in twelve out of sixteen queries (Q3, Q12, Q15, Q18, Q19, Q26, Q27, Q42, Q52, Q55, Q91, Q6), providing fast performances with times ranging from 2.51 seconds to 35.75.

The four remaining queries (Q7, Q43, Q82, Q84), where HAWQ performs better grants position as the fastest tool. In Q43 has the more significant difference, where HAWQ was 2.14x faster. After the three fastest tools, Drill is the tool that follows in terms of overall execution times, being fastest than HAWQ in Q3 and Q15.

Table 13. Interactive query execution time for 50 GB (in seconds).

	Drill	***HAWQ***	***Hive***	***Impala***	***Presto***	***Spark***
Q3	30.35	32.19	33.91	16.27	24.44	39.73
Q7	32.98	7.28	29.48	11.93	26.54	36.96
Q12	17.08	14.19	17.77	4.97	13.18	24.02
Q15	23.26	25.89	21.07	14.12	16.67	26.79
Q18	27.94	25.32	47.03	14.85	19.31	51.89
Q19	33.78	32.96	22.97	8.16	24.98	30.44
Q26	19.28	14.95	27.89	4.21	17.28	35.42
Q27	31.26	29.61	27.34	4.51	25.41	32.09
Q42	16.34	11.77	13.02	2.72	16.95	23.02
Q43	16.04	10.87	32.48	23.32	22.95	40.08
Q52	13.34	3.81	14.63	3.14	17.52	24.14
Q55	13.23	4.33	15.29	2.71	17.61	23.14
Q82	20.15	36.26	50.22	35.75	16.01	55.42

Q84	14.24	7.68	29.29	10.27	9.72	37.35
Q91	18.66	8.66	18.48	8.44	14.37	24.58
Q96	13.14	3.85	27.34	2.57	16.27	37.36
Total	**341.07**	**133.42**	**428.21**	**167.94**	**299.21**	**542.43**
Speedup	**4.07**	**4.07**	**1.27**	**3.23**	**1.81**	**1**

We can see more clearly in Figure 34, although Impala is surpassed by HAWQ when we perform all the queries, it still provides good performances in five queries (Q3, Q18, Q26, Q27 and Q91), more than half this query set. The remaining three queries are performed faster by the fastest tool HAWQ. In the referenced figure, the long times performed by Hive in Q18, Q82 and Q84 draws attention since they have relevant different towards other execution times.

On other cases Hive manages to perform similarly the fastest tools like in Q3, where HAWQ was less than one second faster.

Also on Q7, where Drill was the slowest tool, HAWQ and Impala performed respectively four and two times faster than Hive. The third fastest tool Presto, performed this query with few advantage towards Hive (only three seconds faster). Other example of Hive's performance proximity with the fastest tools can be seen in Q27 (comparing with HAWQ and presto). With the scale of the SF we can observe that are spread between the usual fastest tools, in this query set, our top performer HAWQ was only able to perform the fastest times in Q84.

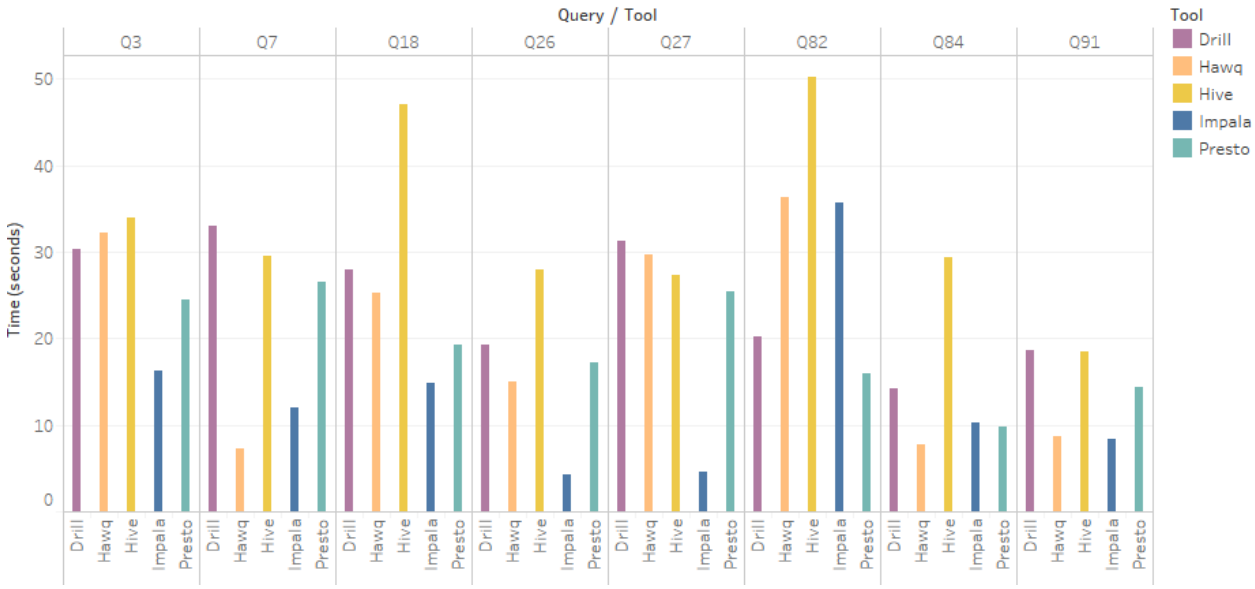


Figure 34. Interactive sample Query Set for 50 GB

In the TPC-H 100 GB benchmark, our cluster resources started to reach its limit and consequently we couldn't run several queries, especially in Presto. For the TPC-DS benchmark, we still did not observed failures on Presto until reaching the current SF. When we reach the 100 GB SF we could not run Q82 on Presto due to the intensive I/O caused by demanding *JOIN* operations. Consequently, as seen before, the amount of data processed by

the query made Presto run out of memory causing query execution failure. From the begining we identified the delay running this query, but we were always able to run the query in all tools.

Like we saw on the 10GB SF Cloudera Impala was again the fastest tool with significant time difference, having a speedup of 3.15 for the current SF. If we consider the total time it took to run all the queries, it leads with a huge time difference from the other tools.

Following Impala, as second fastest tool we have Hive. In the TPC-H and even with in previous SF's of this benchmark, we observe that Hive seemed to catch up the fastest tools as the data volume grows. Here we observe Hive manages to surpass tools that usually were placed in the top three fastest tools (HAWQ and Presto).

In the current scenario we see that Hive surpasses HAWQ and Presto in several queries like Q3, Q7, Q15, among others. Also, when we consider the time required to run all the queries, Hive is always superior.

This is the first time that HAWQ and Presto are in between the slowest tools. The scale of SF had huge impact on their performances, specially on HAWQ, that now presents performances that even the usual slowest tool Spark surpassed. The same happen on Presto, that now was surpassed by Drill, the third fastest tool.

Table 14. Interactive query execution Time for 100 GB (in seconds).

	Drill	***HAWQ***	***Hive***	***Impala***	***Presto***	***Spark***
Q3	35.21	56.17	36.47	29.28	49.21	76.59
Q7	65.81	58.79	35.62	32.65	45.67	98.29
Q12	25.65	34.11	19.37	8.74	17.77	95.16
Q15	39.67	66.72	23.74	19.62	32.76	37.12
Q18	45.25	53.34	65.18	20.17	44.18	71.41
Q19	61.36	74.39	26.97	15.77	47.05	32.97
Q26	45.33	67.61	33.29	7.09	33.01	40.73
Q27	62.58	65.24	28.25	11.14	63.42	38.74
Q42	21.51	59.84	15.84	5.37	43.67	27.88
Q43	25.39	58.43	41.92	46.49	48.94	56.19
Q52	19.42	56.16	16.92	5.67	42.56	27.12
Q55	18.02	56.32	17.09	4.25	38.37	29.65
Q82	54.96	69.53	137.58	101.34	-	121.06
Q84	20.11	16.28	42.72	30.82	13.62	46.81
Q91	22.78	16.23	19.75	11.51	20.47	50.41
Q96	21.68	59.55	38.78	5.91	33.59	49.04
Total (W/Q82)	**529.77**	**799.18**	**461.91**	**254.48**	**574.29**	**778.11**
Speedup	**1.51**	**1**	**1.73**	**3.14**	**1.39**	**1.03**

Although Hive is the second fastest tool for this SF, the first thing that we notice looking at Figure 35, is its performance in Q82. For this query, Hive is much slower than the the tools present in this query set. Similar examples can be observed on Q18 and Q84. We

also observe that the fastest tool Impala performs best with great advantage (mainly on Q18, Q26 and Q27). Other hand, returning to Q82 we see that like HIVE, Impala struggles to perform Q82.

Once again, due to the amount of processed by the query, "spill to disk" is activated, consequently query execution time increased. This query was the only case where we detected the action of this feature.

Regarding HAWQ, Figure 35 illustrates in a clearer way the decrease of performance present in this SF. In this query set there are cases it performed the worst times (Q3, Q26, Q27). On other cases, it still manages to surpass Hive (Q18, Q82, Q85 and Q91). However, there was huge performance degradtation comparing with results that this tool has accustomed us to. Finally, regaring Drill, although it's the third fastest tool, we can see that it struggles to run some queries in this set, like Q7, Q27 and Q91.

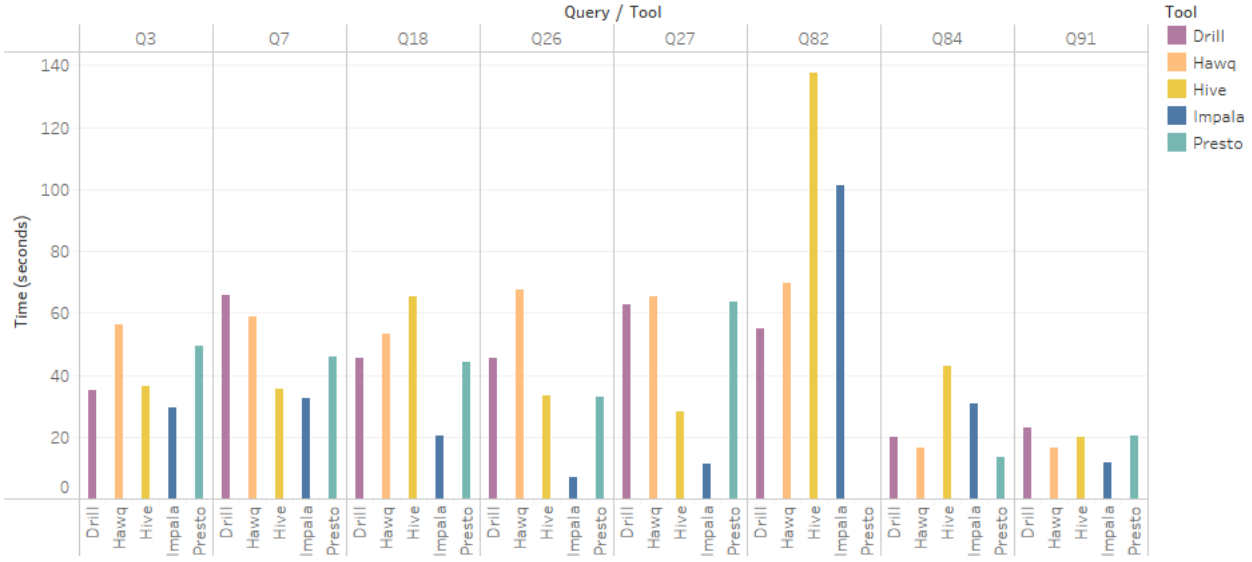


Figure 35. Interactive sample Query Set for 100 GB

6.3.2 Main Findings Over TPC-DS Interactive Queries Results

Analyzing the results obtained we find that Hive and Spark are less suitable when querying in low data volume, like we conclude in the TPC-H benchmark. On 10, 30 and 50 GB SF, we observed that in-memory tools (Impala, HAWQ and Presto) are always able to provide the best results, usually followed by Drill. Figure 36 illustrates in overall in how many queries individual tools managed to achieve the fastest times, here its clear to see that mainly Impala achieved the fastest execution times, specially on the 100GB, where Impala was the fastest on all the 16 queries.

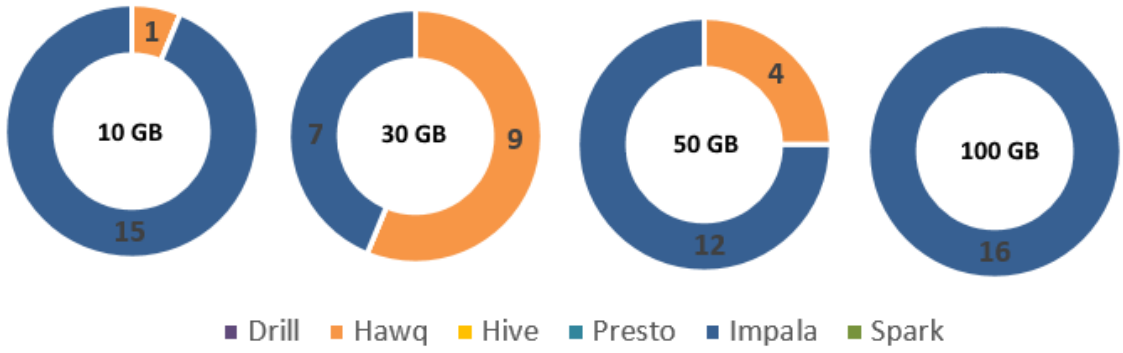


Figure 36. Number of Fastest Times on TPC-DS Interactive Queries by Tool

Again, another fact according to what we have seen in the previous benchmark. We also refered that as we scale the SF, Hive had the tendency to start catch up with the fastest tools, performing times with less significant differences. This could mean that Hive gains advantage as data increases. When we retrieved results for the 100GB TPC-DS Hive surpassed the usual fastest tools Presto and HAWQ, becoming the second fastest tool thus confirming our opinion. For this last SF we also noticed the significant performance decrease on Presto and specially HAWQ.

Nevertheless, we noticed that for the current benchmark, although Presto didn't deliver top performances, it only failed to perform one query in all benchmark (Q82 on the 100GB SF), much less failures than in the previous analyzed TPC-H Benchamark. Regarding Impala, we already noticed that it always assures query completeness, due to the application the "spill to disk" feature (although it extended execution times drasticly).

On the current benchmark, we didn't notice the activation of this feature on the performance of Impala, except for the execution of Q82 when performing queries to the 100 GB SF. On the other hand, the extreme degradation on HAWQ on the last SF, can be explained by the activation of "spill to disk". In this case HAWQ was gravely impared, unlike what we observed on the TPC-H. In the TPC-H HAWQ seemed to manage the I/O to disk more efficiently than Impala, here we observed the opposite. As identified earlier, SQL clauses that more require memory allocation that could activate the spilling mechanism include GROUP BY, JOIN operations over large tables and sorting operations (*ORDER BY* and *GROUP BY*). This tendency maintained in the current benchmark.

In a nutshell, we saw that Hive is surpassed by all tools in low data volume data volumes, being surpassed by all other tools, except for spark, that maintained as the slowest tool. We maintained our opinion that in-memory tools like Impala, Presto and HAWQ deliver faster execution times and overall better performance. Still, when we reach 100 GB for this SF, the only in-processing tool that stands out is Impala.

Presto and HAWQ lose their edge, being surpassed by Hive. This means that is it's very likely that Hive can keep up with in-memory processing tools, or at least perform without significant delay towards them. Since Hive is constantly being developed, we believe that it can reach a point that it can not only provide batch processing, but aso interactive queries.

Nevertheless, while this possibility doesn't become a certainty, as previously stated, the scale of cluster resources, mainly RAM could be the game changer, assuring the superiority of in-memory processing.

6.3.3 Reporting queries

TPC-DS reporting queries capture the reporting nature of a decision support ystem. They include queries that are executed periodically to answer well-known, pre-defined questions about the financial and operational health of a business.

Unfortunatly for this subset of the benchmark it wasn't possible to include HAWQ, one of the best performers of our evaluation, due to the lack of availability of our cluster. Still we present the reporting queries subset for the remaining tools. Although reporting queries tend to be static, minor changes are common. From one use of a given reporting query to the next, a user might choose to shift focus by varying a date range, geographic location or a brand name [58].

These queries involve multiple fact tables or large intermediate datasets, being much more complex to evaluate than the previous. They include several nested subqueries and JOIN operations (there are queries that perform subqueries, *JOIN* and *UNION* operations with tables more than one time in the same query). In Table 15, we map the tables involve in each query, to complement this map please refer to Appendix H, where we present all the ANSI SQL reporting queries.

Table 15. Table mapping by reporting query.

	Q 17	***Q 21***	***Q 32***	***Q 40***	***Q 46***	***Q 58***	***Q 68***	***Q 76***	***Q 79***	***Q 88***	***Q 90***	***Q 92***	***Q 93***	***Q 95***	***Q 97***
Item	***X***	***X***		***X***		***X***		***X***							
Catalog_Page															
Call_Center															
Catalog_Returns (fact)				***X***											
Catalog_Sales (fact)	***X***		***X***	***X***		***X***		***X***				***X***			***X***
Customer															
Customer_Address					***X***		***X***							***X***	
Customer_ Demographics															
Date_Dim	***X***	***X***	***X***	***X***	***X***	***X***	***X***	***X***	***X***					***X***	
Household_ Demographics					***X***		***X***		***X***	***X***	***X***	***X***			
Income_Band															
Inventory (fact)		***X***													
Promotion															
Reason													***X***		
Ship_Mode															
Store	***X***				***X***		***X***		***X***	***X***					
Store_Returns (fact)	***X***												***X***		
Store_Sales (fact)	***X***				***X***	***X***	***X***	***X***	***X***	***X***		***X***	***X***		***X***
Time_Dim										***X***	***X***				

Warehouse		***X***		***X***										
Web_Page											***X***				
Web_Returns (fact)															
Web_Sales (fact)											**X**			**X**	
Web_Site														**X**	

The results fot the 10GB scale factor are presented in Table 16, from a general view, these type queries result in higher execution times, independently of the tools. For TPC-DS interative queries on the same 10GB SF, taking Impala as example, the most part of the queries have query execution times under five seconds, providing nearly instant result retrieval. We were expecting these differences since reporting queries have higher complexity.

Still, in overall Impala was the fastest tool for this SF, beating Presto, the second fastest tool in every query except for Q95.

Analyzing execution times, this query that seems more complex since it has the longest query execution times, regardeless the tool. Impala was running most of the queries under ten seconds, but in this case, we see it required much more, performing Q95 in 44.39 seconds.

We also observe that the third fastest tool Drill could not run this query. We performed some queries with high level of complexity in the previous presented results, but this was the first time that Drill couldn't perform.

Q95 produce a count of web sales and total shipping cost and net profit in a given sixty day period to customers in a given state from a named web site for returned orders shipped from more than one warehouse.

In order to do this, it performs several JOIN operations, including the fact table Web_Sales and the dimensions Customer_Address, Web_Site and Web Returns (retrieving 9579959 rows). After performing these the final results are presented to the user through aggregate functions, more specificly, two SUM functions and a COUNT DISTINCT.

The third fastest tool Drill, in terms of performance, was beaten by Presto in every query, with exception of Q21. While Presto performed the query in almost twenty-four seconds, Drill surpassed it, retrieving results in nearly fourteen seconds.

Although Presto is second fastest tool, Impala presented performances with significant difference. Presto was much more slower, specially in queries like Q21, Q32, Q58 and Q88, resulting in a speedup of 2.72, against the 6.89 speedup of Impala, meaning its near seven times faster than Spark.

Table 16. Deep Reporting query execution time for 10 GB (in seconds).

	Drill	***Hive***	***Impala***	***Presto***	***Spark***
Q17	31.61	34.71	21.81	23.84	50.66
Q21	13.82	17.39	5.07	28.91	24.39
Q32	20.32	14.06	2.61	10.61	22.56
Q40	27.47	24.12	4.92	14.06	31.72
Q46	9.59	20.66	9.11	9.01	22.62
Q58	61.91	51.23	7.54	20.83	53.96
Q68	11.99	15.46	3.29	8.22	23.27

Q76	31.49	35.45	3.01	13.36	43.48
Q79	16.09	18.74	2.66	8.61	22.78
Q88	45.12	73.21	4.02	28.82	71.91
Q90	15.18	26.71	4.34	8.72	32.27
Q92	21.25	38.78	3.13	10.39	52.83
Q93	29.69	33.72	5.77	17.62	49.83
Q95	-	84.56	44.39	28.37	99.68
Q97	17.81	39.45	3.29	10.91	52.63
Total (W/Q95)	**353.34**	**443.69**	**80.57**	**203.52**	**554.91**
Speedup	***1.57***	***1.25***	***6.89***	***2.72***	***1***

From the chosen reporting queries set presented on Figure 39, Q17 is the one only one that uses three fact tables, involving store_sales, store_returns and catalog_sales, also involving the dimensions, date_dim (three times to apply different filters), store and item, as we can see in. After retrieve heavily filtered results (3114 rows), the *GROUP BY* is applied, resulting in 1557, on which several aggregations (including several *stddev_samp*, *COUNT*, *AVG* and custom calculations) are performed, but only 100 are restrieved to the user (*TOP* 100).

As usual Spark kept as the slowest, followed by Hive that was slower than Drill only by three seconds. Presto and Impala were the fastest tools presenting similar performances. As we referenced above, Q17 and Q95 requires above average execution time, the significant difference towards the remaining queries is clearer in the figure below.

We also noticed that for Q46, Drill had pratically the same performance as Presto and Impala, with fewer miliseconds of difference. This query contains a subquery as a nested SELECT statement in the FROM clause of an outer SELECT statement. This nested select perform joins over five tables, including the biggest fact table store_sales, involving 82358 rows.

After the filtering, the result includes two aggregation SUM functions are made with a *GROUP BY*, reducing the result set to 37812 rows, only 100 are presented to the user. Although this query processes a significant amount of data, Drill performed similarly to the fastest tools.

In most queries, Presto was able to be the second fastest tool, although in Q21, Drill and Hive managed to be faster with some difference.

This query joins data from the dimensions inventory, warehouse, item and date_dim, processing a total of 2768272 rows.

After the retrieval of rows, the query performs two SUM's that nests CASE expressions that evaluate the *CAST* of the text column d_date from the dimenstion date_dim as date, reducing the rowset to 1384136, finally the final filtering stage in the WHERE clause filters 692068 (only the top 100 are retrieved to the user). This kind of operations are performed on other queries, but in this case, there is much more data to be processed, leading to the degradation of Presto performance.

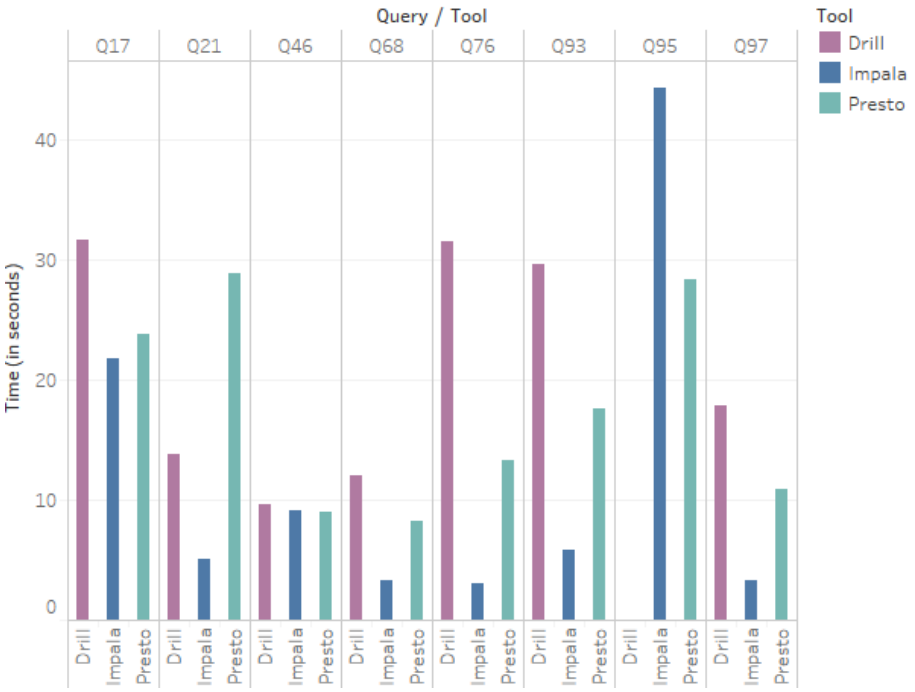


Figure 37. Reporting sample query set for 10 GB.

For the 30GB SF presented in Table 17, Impala maintained a secure advantage towards the remain tools. We also notice that with the growth to the 30 GB SF, Presto is no longer capable of running the demanding Q95.

Presto and Drill kept their performance as second and third fastest tools. Although Drill in overall requires more time to perform all the queries, Presto seems to perform similarly or with few difference, e.g. Q17, Q21, Q40 or Q46.

On some cases Drill manages to surpass Presto, in cases like Q21, Q46, Q68, among others. On the other hand, like in the previous SF, Drill seems to take abnormal time to execute Q58. In this query, besides the heavy JOINS performed involving the fact table catalog_sales, dimensions date_dim and item, it also performed numerous arithmetic calculations to filter information on the *WHERE* clause. The several fields returned by the query also results from other arithmetic calculations in the *SELECT* statement. Drill needed more than 100 seconds, even Hive and Spark performed better.

Regarding Q95, it keeps presenting huge execution times for the tools that are able to run it. We saw that our leading tool, Impala, struggles to perform this query and on this SF this is clearer. The amount processed made Impala run out of memory, and then the "spill to disk" feature extended execution time, being only possible to retrieve results after 159 seconds. Once more we observe the performance degradation caused by this feature, resulting in performances similar to the ones provided by the slowest tools. Hive was about 3 seconds slower and Spark, about 6 seconds slower.

Table 17. Deep Reporting query execution time for 30 GB (in seconds).

	Drill	***Hive***	***Impala***	***Presto***	***Spark***

Q17	43.81	60.11	32.43	43.01	63.27
Q21	8.16	18.93	9.33	10.32	29.02
Q32	26.97	23.82	3.37	15.01	32.16
Q40	33.28	33.21	7.29	29.54	40.34
Q46	12.89	25.11	12.97	15.24	28.42
Q58	103.55	83.08	12.15	32.41	74.76
Q68	11.39	20.06	7.09	14.19	24.97
Q76	39.78	54.03	4.34	16.81	68.62
Q79	26.72	27.26	3.09	14.61	33.98
Q88	70.02	100.81	7.05	48.65	98.06
Q90	18.94	33.75	5.04	13.98	41.94
Q92	30.35	80.97	7.72	15.26	89.93
Q93	46.97	156.94	9.62	32.09	168.36
Q95	-	162.61	159.13	-	165.31
Q97	29.89	81.65	8.54	14.37	92.42
Total (w/Q95)	**502.72**	**803.82**	**130.03**	**315.49**	**886.25**
Speedup	**1.76**	**1.10**	**6.82**	**2.81**	**1**

The set of queries of Figure 38, leaving Spark aside, shows us that Hive is the overall slowest tool, it could keep up with the fastest tools, running queries with significant execution times differences.

Although we referenced above that Drill can keep up and even surpass Impala and Presto in cases like Q21 and Q46. We can see in the figure below that there are also cases where Drill performs similarly to Hive (Q32, Q40). Looking at Hive, on this query set, it always the tool that required more time to perform the with significant difference. The only exceptions are on Q58 and Q32, where Drill took more time to perform.

We also can see that for this query set, Impala has significant advantages over Presto, being in Q21 the smaller time difference (less than one second). On all other cases, Presto presented much longer execution times.

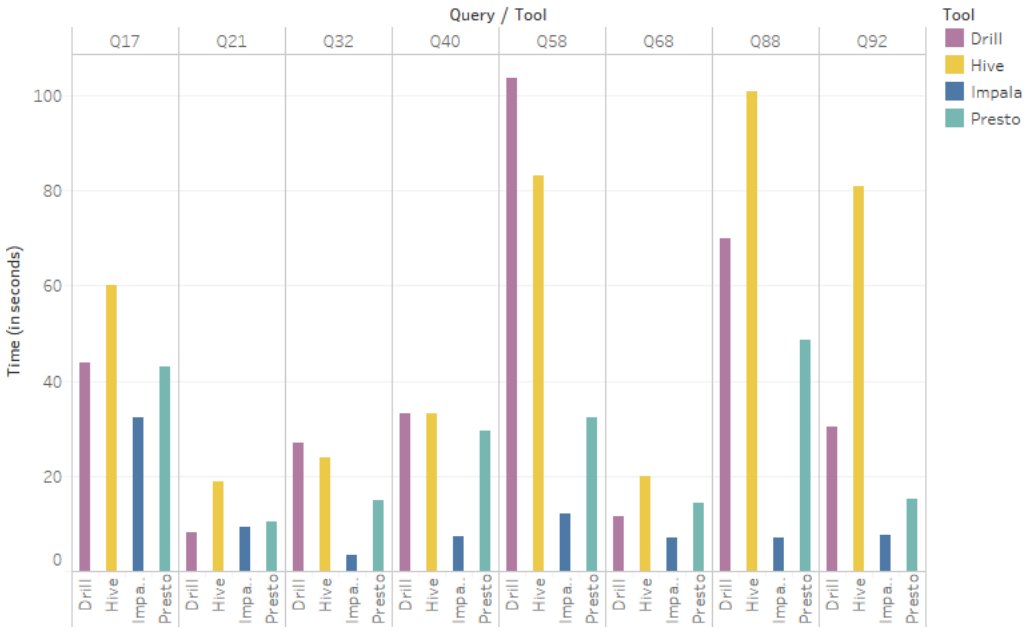


Figure 38. Deep Reporting sample query set for 30GB.

The results for for the 50GB SF are presented on Table 18, where we can see the growth of data volume clearly extends the required time to run queries. Our slowest tools, Hive and Spark now surpass the barrier of the 1000 seconds (around seventeen minutes).

Impala keeps presenting better performances in all queries, retrieving results in acceptable times, except in Q95 that required 274.35 seconds due to the delay caused by the amount of data and the "spill to disk" feature.

Presto also maintained its position as second fastest tool, requiring 490.67 seconds to perform all the queries, almost four times more than than Impala.

Third and fourth positions in terms of performances stays the same with Drill beating Hive in all queries except in Q58, Q76 and Q79.

Table 18. Deep Reporting query execution time for 50 GB (in seconds).

	Drill	***Hive***	***Impala***	***Presto***	***Spark***
Q17	51.23	76.61	37.49	54.48	76.41
Q21	13.85	21.53	13.27	17.95	31.56
Q32	21.07	28.56	5.57	21.81	36.51
Q40	38.52	43.12	8.07	33.19	56.75
Q46	12.28	28.61	13.62	22.81	34.91
Q58	139.02	107.78	17.27	51.11	119.01
Q68	14.41	23.36	12.58	22.77	30.92
Q76	69.24	63.77	6.09	32.96	74.76
Q79	35.84	34.25	6.02	21.72	38.74
Q88	78.49	128.29	12.77	81.04	127.74
Q90	22.25	40.15	6.26	18.15	46.16
Q92	41.92	110.59	12.59	30.01	128.22
Q93	58.68	265.46	12.27	56.76	267.68
Q95	-	238.89	274.35	-	241.29
Q97	50.39	129.17	15.81	25.91	134.98

Total (w/Q95)	647.19	1101.25	179.88	490.67	1204.35
Speedup	1.86	1.09	6.70	2.45	1

Figure 39 presents another query set for all the tools analyzed, where we can see in a clearer way that Impala keeps having a significant time advantage over Presto in most queries, specially in Q32 and Q40 where quickest tool performed the query four times faster.

Although there are several queries where Impala surpasses largely Presto, its in Q88 that the biggest time difference is present, where Impala performed the query about six times faster. In overall, Impala was 2.73 times faster than Presto.

Regarding Drill, we see that Q58 has the longest execution times of all the query set, consequently as referenced previously, it was surpassed largely by the remaining tool, including the slowest tools.

We also notice that like in the previous SF, in some queries Hive and Spark perform most queries with few time difference, this is noticeable when looking at Q17, Q58, Q88 and Q92.

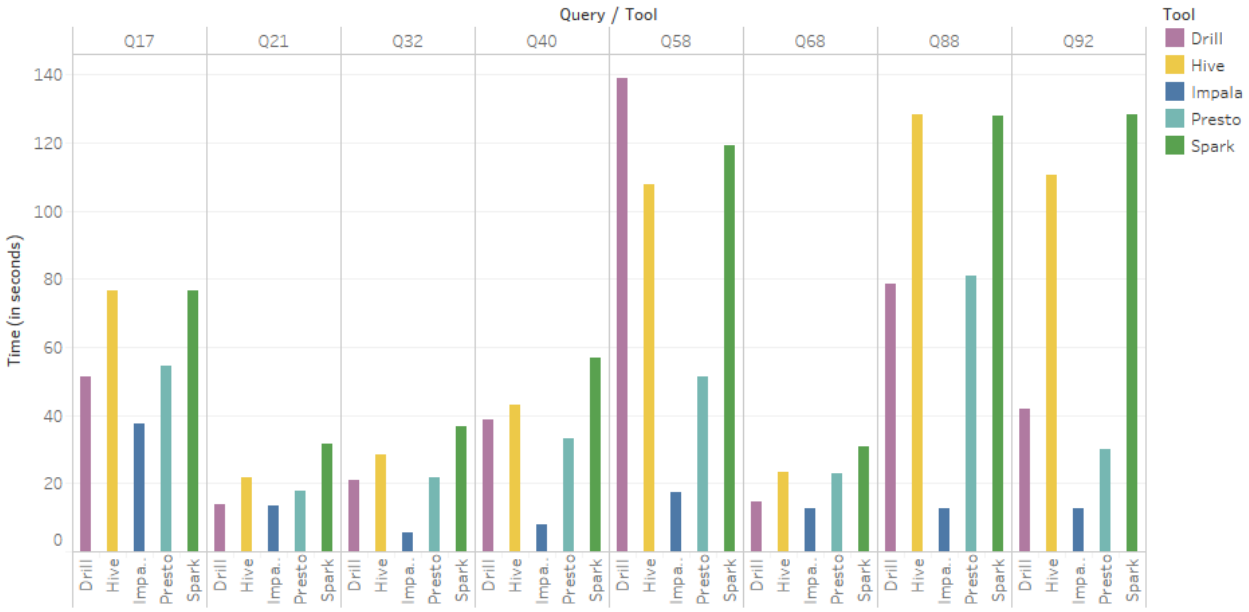


Figure 39 Deep Reporting sample query set for 50GB.

For the 100GB SF, Impala was the only tool that maintain performance, being again the fastest and stable tool in terms of query execution time. Although Presto maintained as the second fastest tool, we can see that query execution times raised considerably, most queries execution times are around 60 seconds and for some cases surpassing 100 seconds.

For the current SF, we see the third fastest tool Drill surpassing Presto more often. Drill manages to surpass Presto in seven out of fifteen. Still in overall, considering the time required to run all the queries, Presto was faster by 83 seconds, since it was able to perform some queries significantly faster (like Q21, Q58, Q76, among others).

In general terms we must say that once again this volume of data stretched queries execution times in all tools, consequently we now see that Presto, Drill, Hive and Spark required more than 1000 seconds to run all the queries.

Even the lead tool Impala, that in the previous SF was still capable to present reasonable times on all queries (except in the demanding Q95), suffered visible performance degradation.

Regarding the most demanding query, Q95, when running in Impala the execution time goes above 1000 seconds.

This kind of performance by Impala was only showed in the execution of TPC-H most demanding query (Q21). As referenced, the excessive amount of data processed by the query, doe not fit in our Impala's memory, triggering the "spill to disk", causing data write in disk, extending greatly query execution time.

Table 19. Deep Reporting query execution time for 100 GB (in seconds).

	Drill	***Hive***	***Impala***	***Presto***	***Spark***
Q17	118.88	137.75	66.11	104.25	179.38
Q21	25.27	29.78	18.92	73.65	48.46
Q32	33.89	36.98	9.23	35.93	50.47
Q40	52.42	52.54	13.46	68.19	61.71
Q46	21.68	34.14	25.47	63.83	42.05
Q58	290.32	123.29	35.88	131.38	134.41
Q68	16.07	30.55	19.49	64.56	37.98
Q76	78.91	82.34	9.81	46.48	88.32
Q79	51.03	46.86	9.89	63.81	55.63
Q88	132.95	163.63	19.29	155.83	158.32
Q90	27.25	62.36	7.15	23.88	62.89
Q92	67.81	136.82	30.45	56.83	150.02
Q93	139.78	395.28	26.66	92.29	323.79
Q95	-	396.01	1087.78	-	371.46
Q97	65.41	154.66	36.22	57.51	153.82
Total (w/Q95)	**1121.67**	**1486.98**	**328.03**	**1038.42**	**1547.25**
Speedup	**1.37**	**1.04**	**4.72**	**1.49**	**1**

In Figure 40 we observe that for the present query set Impala keeps providing the best performances, being only surpassed by Drill in Q68. As previsouly stated, for this SF, although maintaing as second fastest tool, Presto presents an increase of execution times. Analyzing time performed in Q21, Q40, Q46 and Q68 we can see that Presto was the slowest tool, being surpassed by Drill and Hive.

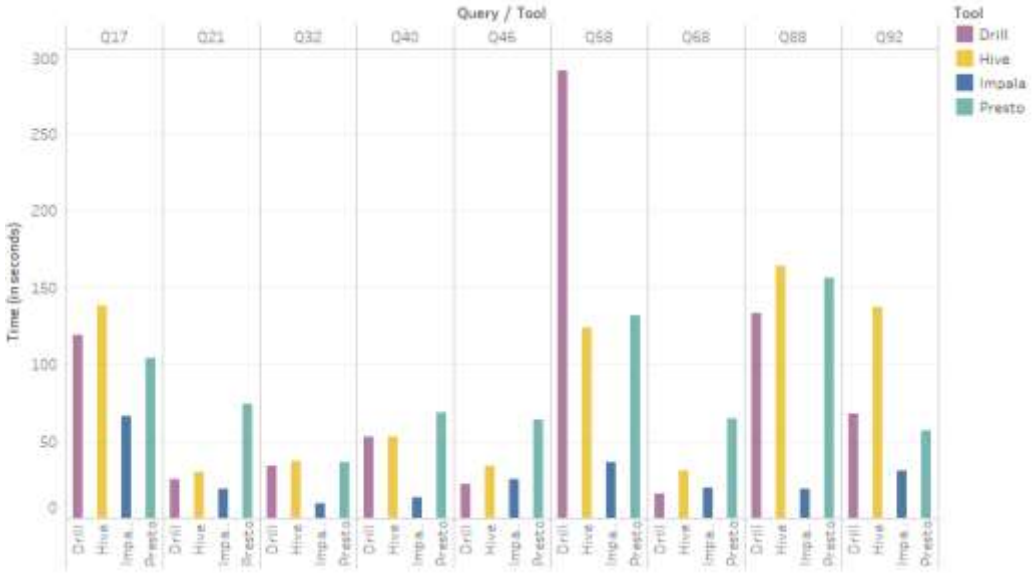


Figure 40. Deep Reporting sample query set for 100GB.

6.3.4 Main Findings Over TPC-DS Reporting Queries Results

We perform a lighter and less detailed analysis regarding the reporting queries, just to observe how the evaluated tools behave processing of this kind and evaluate if the tools that managed to be faster keep an also outperforming others.

These queries involve multiple fact tables or large intermediate datasets, including intense arithmetic calculation and severeal nested subqueries, being much more complex to evaluate than the previous. Still, as presented in previous benchmarks, the in-memory processing tools, Presto and specially Impala kept as top performers, followed by Drill. Although Presto achieves the second fastest query execution time, we observe that in general it takes much more than Impala in most of the queries.

As referenced previously, it was not possible to evaluate HAWQ performance. Figure 41 illustrates in overall for how many queries the tools have achieved the best times, showing that Impala achieved the fastest execution times in most of the queries (for all the evaluated SF, Impala was the fastest on 13 out of 15 queries).

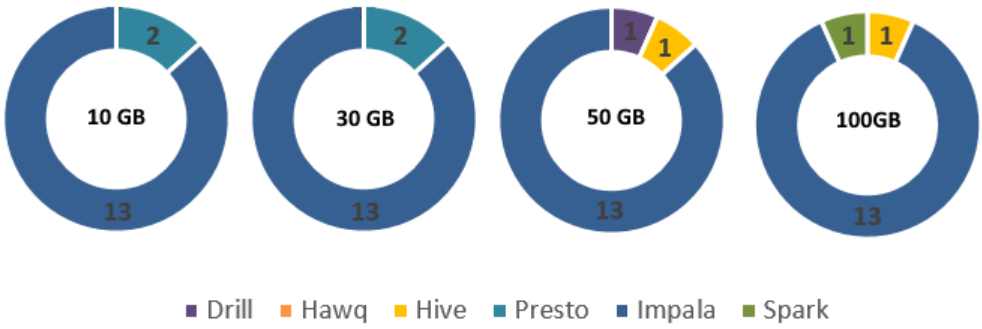


Figure 41. Number of Fastest Times on TPC-DS Reporting Queries by Tool

Taking into account that usually HAWQ was between the top performers, especially on lower SF's, we predict that it would be capable to surpass Drill.

Nevertheless, Impala was the fastest tool on all scale factors. Although the evaluated queries structure is more complex and difficult to read, due to its several nested subqueries and numerous JOIN operations, we notice that the "spill to disk" feature was rarely detected.

There were only two cases on the 100GB SF where we notice clearly the delay in execution times, on Q17 and pricipally Q95, where execution time surpassed 1000 seconds. One more reason to predict that HAWQ would perform as one of the fastest tools.

The absence of the "spill to disk" made Impala provide very fast execution times, supporting our opinion that with higher amounts of RAM memory, in-memory processing tools would always provide better performance, supporting low latency and interactive queries.

7 Conclusions and Future Work

In this chapter we present the conclusions and future workm, providing an overview of the problem statement. We also describe our main research contributions and a summarized analysis of the results obtained. We end by discussing the main limitations of our work and directions for further research.

7.1 Overview of the problem statement

This work focuses on the study of Big Data querying tools, describing and comparing their main characteristics to identify the ones that can perform ad hoc queries on huge datasets.

We started by providing an overview of the Big Data concept, although nowadays is a relatively well-known definition, it still faces a lack of consensus and rigor in its definition and standardization. Consequently, common users associate the Big Data concept to complexity and hugely learning efforts.

Due to this fact, we outlined the conceptual framework for Big Data, where we provided a set of concepts and notions that in our opinion are provide a knowledge base of the area, so even less experienced users can understand. We also present a technological framework, where we presented technological concepts in which we highlight Hadoop and its capabilities to to store and handle huge amounts of (unstructured) data. The powerful and ever-growing Hadoop ecosystem is, and has been, a clear leader in operational and exploratory data computation and analytics.

In order to open up Big Data Analysis to a wider public, we see the increase of SQL-on-Hadoop systems as an opportunity, as many users are familiar and comfortable with the use of Structured Query Language (SQL). Thus, this concept opens the Big Data World to a wider audience, there is a need to identify clearly Big Data Querying tools that provide SQL-on-Hadoop capabilities and evaluate if they are capable of provide fast interactive queries.

Analyzing the state of the art Big Data Querying tools, we chose to evaluate the tools Drill, HAWQ, Hive, Impala, Presto and Spark, describing their architectures and main characteristics.

Despite the existence of some work in the Big Data querying and processing, there is a lack of studies, analyzing scalability evaluation and comparing query execution time, specifying the ideal scenario to use the tools in a clear way. In order to find if the selected set of tools can deliver fast performances and execute *ad hoc* queries within seconds even at scale and complement theoretical research on these tools, we performed an experimental

evaluation and result analysis of Big Data analytical tools performances using TPC-H and TPC-DS benchmarks.

7.2 Research contributions

This work evaluated the state of the art Big Data querying tools and as result it was possible to publish a paper (Appendix A) on WorldCIST'17 5th World Conference on Information Systems and Technologies. This work received an invitation to submit an extended version (Appendix B).

Finally, we submitted a paper intitled "Experimental evaluation of Big Data Analytical Tools" to CAISE'18 (Appendix C). The decision on the publication two works is still pending.

In order to perform experimental approaches, we had to prepare a close to a real case scenario environment (Cluster) and configuring it, in order to install Hadoop Distributions to support the selected querying tools. This process consumed a fair amount of time. We usually observe in related works the characteristics of the environments used, but we don't see details regarding the preparation of a suited environment to process Big Data. Due to this, our work also includes the description of the deployment process and configurations of a cluster, also detailing how install and used the selected Big Data Querying to perform the described experiences. This level of detail on the deployment and configuration was made aiming to enlighten more inexperienced users. Consequently, this work can be use as guideline for users that like us, don't have experience in the preparation of these type of enviroments and configuration of this kind of tools.

7.3 Results Obtained

Big Data processing tools can deal with the massive quantity of data that exists nowadays, allowing users to perform ad-hoc querying, after performing the referred benchmarks we can consider that there is no one-size fits all SQL-on-Hadoop tool.

The results provided by the performed benchmarks, revealed that in overall the in-memory processing tools Impala, HAWQ and Presto provided the best performances. Especially when we were dealing with lower scale factors, these tools were capable to provide very fast query execution times, consequently, being the most suited options to adhoc queries and interactive analysis.

Using in-memory processing, all the relevant data processed by the is loaded into RAM, avoiding side steps, unnecessary I/O and latency, allowing faster response times. Results showed that in-memory processing tools, specially Impala, HAWQ and Presto, are very efficient if the processed data fits in the available memory.

Nevertheless, when we scale the data, specially on the 100GB SF, we observed that performance suffers a significant degradation. Regarding Presto, we often saw that it was not

able to complete some of the queries, when one of the nodes ran out of memory, query execution stoped imediatly.

Unlike Presto, Impala and HAWQ were able to execute all the queries even if they ran out of memory, since they have activated by default the "spill to disk" feature. This feature prevented queries that use memory-intensive operations from failing with out-of-memory errors. In those cases, data is written in disk avoiding query to crash, although it greatly impairs the performance. Consequently, as we saw in the presented results, in-memory processing tools stops being the top performers, being surpassed by slowest tools.

After Impala, HAWQ and Presto, Drill usually provided the reasonable results, being in most cases ahead of Hive and Spark.

Still, when we performed queries in Impala and mainly HAWQ that process big anounts of data, triggering the "spill to disk" activation, Drill usually outperform them. Regarding Hive, that we see as standard, the first to support SQL-on-Hadoop and it is included in all Hadoop distributions.

After analyzing the obtained results, we found that Hive is less suitable for querying in low data volumes, due to the long coordination time of jobs, but if we look at the different SF, as the data volume grows, Hive improves its performance and seem to catch up with the fastest tools. Particularly, in these cases, when the volume of data starts to increase, the impact of the initial coordination time is less significant, and Hive can achieve better results. In sum, Hive's robustness start to emerge when the datasets start exceeding the amount of RAM in the cluster, demonstrating better performance than systems designed for interactive querying. On other hand, in no circumstance it was able to surpass Cloudera Impala.

Spark was the slowest of the selected set of tools. We were surprised by the times obtained in the benchmarks, since like the fastests tools evaluated, it uses in-memory processing. For Spark in specific, in-memory processing didn't provide good perfomances, which lead us to believe that Spark its more suitable when the objective is not just to query data, but perform algorithmic analysis, statistics and Machine Learning. We believe that to improve Spark, we should not just use SQL, but take advantage of other SparkSQL features, like dataframes. In order to do this, we would have to build scripts, using Scala or other programmatic language supported and include the queries. This is a much more complex and time-consuming process, implying much more that running a simple SQL query, like we did in the remaining tools, and so we conclude that Spark general-purpose is not to be SQL layer for interactive/exploratory analysis.

In general, and in most of the cases, tools that can support *ad hoc* and interactive analysis are Impala, Presto and HAWQ, the in-memory processing allowed them to perform as the fastest tools in most of the scenarios. Mainly on Impala, we observed many cases where it is possible to nearly instant result retrieval. Of course, when we scale the data, the degradation of performance caused by the "spill to disk" is relevant. Nevertheless, in a real case scenario, where clusters have much higher amount of RAM, it would be assured that data would fit in memory, and so, this feature wouldn't be activated. This would result in performance improvements for all the tools, but mainly in the ones that use in-memory

processing. Consequently, we think that Impala and Presto would perform as fastest tools, followed by HAWQ and Presto.

7.4 Limitations

We now reflect on the limitations of our work. Firstly, in an initial stage of the research, we felt difficulty to focus on the selection Big Data querying tools, since there is a vast amount of Hadoop components and technologies associated. Another difficulty was arranging the infrastructure to deploy an Hadoop environment. The first contact with this kind of tools was made using only a personal laptop, which was unfeasible. We spent considerable amount of time until finding INCD, that kindly ceded resources to support distributed parallel processing. After this, and using CentOS Linux 7 as the operating system, there was an additional effort of learning terminal commands and syntax and how to perform the configurations required configurations (like the network configuration between nodes, passwordless SSH, etc.).

Regarding infrastructures limitation, our cluster had only four nodes, with sixteen gigabytes of RAM on each node. As seen frequently seen in the literature, this kind of experiments needs significant amounts of RAM and high number of nodes, recalling that this cluster was not the most suited, and so, when we scale data we see exaggerated execution times (mainly in in-memory processing tools like Impala and HAWQ) and query execution failures (mainly in Presto). When the workloads increased, the available memory for each node was enough for implementing some join operations. However, these kinds of tools would benefit from the usage of more RAM memory availability.

7.5 Directions for Future Research

As future work we propose to extend the TPC-H and TPC-DS benchmark to higher scale factors to perform a deeper experimental evaluation of the selected Big Data querying tools in a more suited cluster, meaning, with higher number of nodes and RAM memory.

In these conditions, it would be interesting to evaluate if in-memory processing tools, mainly Impala, Presto and Hawq keep providing the best performances. Also, since Hive is in constant development and seemed able to catch up with fastest tools in some cases, we should be alert for major current developments. Regarding this subject, we realized that Hortonworks has recently launched a new version of its Hadoop distributions (HDP 2.6).

The community seemed to debate a lot and expecting this new version due general availability of a stable version of Apache Hive with LLAP.

Hive's recent LLAP feature was also not evaluated in our experiences since this feature was very recent and still designated as a technical preview in the used Hadoop Distribution HDP 2.5. Since in our experiences Hive seemed to catch up with the fastest tools and this new feature promesses to make Hive queries much more interactive and faster, this could be a game changer for Hive's performance.

References

[1] A. Floratou, U. F. Minhas, and F. Ozcan, “Sql-on-hadoop: Full circle back to shared-nothing database architectures,” *Proc. VLDB Endow.*, vol. 7, no. 12, pp. 1295–1306, 2014.

[2] E. G. Ularu, F. C. Puican, A. Apostu, and M. Velicanu, “Perspectives on Big Data and Big Data Analytics,” *Database Syst. J.*, vol. III, no. 4, pp. 3–14, 2012.

[3] A. Gandomi and M. Haider, “Beyond the hype : Big data concepts , methods , and analytics,” *Int. J. Inf. Manage.*, vol. 35, pp. 137–144, 2015.

[4] C. Costa, “BASIS : Uma Arquitetura de Big Data para Smart Cities,” Universidade do Minho, 2015.

[5] M. Chen, S. Mao, and Y. Liu, “Big data: A survey,” *Mob. Networks Appl.*, vol. 19, no. 2, pp. 171–209, 2014.

[6] George Firican, “The 10 Vs of Big Data,” 2017. [Online]. Available: https://tdwi.org/articles/2017/02/08/10-vs-of-big-data.aspx.

[7] A. DeVan, “The 7 V’s of Big Data,” 2016. [Online]. Available: https://www.impactradius.com/blog/7-vs-big-data/.

[8] R. Maheshwari, “3 V’s or 7 V’s - What’s the Value of Big Data?,” 2015. [Online]. Available: https://www.linkedin.com/pulse/3-vs-7-whats-value-big-data-rajiv-maheshwari/.

[9] B. Y. Mike, “Big Data and Business Intelligence : a data-driven strategy for e-commerce organizations in the hotel industry Big Data and Business Intelligence : a data-driven strategy for e-commerce organizations in the hotel industry,” p. 57, 2015.

[10] C. Costa and M. Y. Santos, “Big Data : State-of-the-art Concepts , Techniques , Technologies , Modeling Approaches and Research Challenges,” *IAENG Int. J. Comput. Sci.*, vol. 44, no. 3, pp. 285–301, 2017.

[11] R. Pandya, V. Sawant, N. Mendjoge, and M. D ’silva 4, “Big Data Vs Traditional Data,” vol. 3, no. X, pp. 192–196, 2015.

[12] P. N. Amundrud, “Opportunities for Automation, Internet of Things, Big Data Analytics and 3D Printing within Oil and Gas Drilling, Production and Transport.,” no. June, 2009.

[13] A. C. Gessler, “MapReduce to Couple a Bio-mechanical and a Systems-biological Simulation,” no. 156, 2014.

[14] S. Mehta and V. Mehta, “Hadoop Ecosystem : An Introduction,” vol. 5, no. 6, pp. 557–562, 2016.

[15] B. PROFFITT, “Hadoop: What It Is And How It Works,” 2013. [Online]. Available: https://readwrite.com/2013/05/23/hadoop-what-it-is-and-how-it-works/.

[16] TutorialsPoint, "Hadoop - MapReduce," 2017. [Online]. Available: https://www.tutorialspoint.com/hadoop/hadoop_mapreduce.htm.

[17] S. Ahmed, "Evaluating Presto as an SQL on Hadoop Solution Evaluating Presto as an SQL on Hadoop solution: A Case at Truecaller," 2016.

[18] K. Krishnan, *Data Warehousing in the Age of Big Data*. 2013.

[19] A. Oussous, F. Z. Benjelloun, A. Ait Lahcen, and S. Belfkih, "Big Data technologies: A survey," *J. King Saud Univ. - Comput. Inf. Sci.*, 2017.

[20] "Cloudera vs. Hortonworks vs. MapR - Hadoop Distribution Comparison," 2016. [Online]. Available: https://www.dezyre.com/article/cloudera-vs-hortonworks-vs-mapr-hadoop-distribution-comparison-/190.

[21] Cloudera, "CDH: The Most Popular Open Source Distribution of Apache Hadoop," 2017. [Online]. Available: cdh: The Most Popular Open Source Distribution of Apache Hadoop.

[22] "Benchmarking Big Data SQL Frame- works," 2016.

[23] J. Bernardino and V. Abramova, "No Experimental Evaluation of NoSQL Databases," *Int. J. Database Manag. Syst.*, vol. 6, pp. 1–16, 2014.

[24] R. J. de S. Marques, "Big data model with SQL support for performance management in telecommunication networks," University of Aveiro, 2014.

[25] T. T. Maposa and M. Sethi, "SQL-on-Hadoop : The Most Probable Future in Big Data Analytics," *Adv. Comput. Sci. Inf. Technol.*, vol. 2, no. 13, pp. 9–14, 2015.

[26] B. Saha, H. Shah, S. Seth, G. Vijayaraghavan, A. Murthy, and C. Curino, "Apache Tez," *Proc. 2015 ACM SIGMOD Int. Conf. Manag. Data - SIGMOD '15*, pp. 1357–1369, 2015.

[27] S. Van Wouw, J. Viña, A. Iosup, and D. Epema, "An Empirical Performance Evaluation of Distributed SQL Query Engines: Extended Report," 2014.

[28] C. Owl, "The SQL on Hadoop landscape: An overview (Part I)," 2015. [Online]. Available: http://cleverowl.uk/2015/11/19/the-sql-on-hadoop-landscape-an-overview-part-i/.

[29] C. Owl, "The SQL on Hadoop landscape: An overview (Part II)," 2015. [Online]. Available: http://cleverowl.uk/2015/12/25/the-sql-on-hadoop-landscape-an-overview-part-ii/.

[30] D. K. Devadutta Ghat, David Rorke, "New SQL Benchmarks: Apache Impala (incubating) Uniquely Delivers Analytic Database Performance," 2016. [Online]. Available: https://blog.cloudera.com/blog/2016/02/new-sql-benchmarks-apache-impala-incubating-2-3-uniquely-delivers-analytic-database-performance/.

[31] S. Sakr, "A brief comparative perspective on SQL access for Hadoop," in *Recent Advances in Information Systems and Technologies, Volume 1*, 2014, pp. 1–9.

[32] M. Kornacker *et al.*, "Impala: A Modern, Open-Source SQL Engine for Hadoop," *CIDR(Conference Innov. Data Syst. Res.)*, 2015.

[33] B. R. Prasad and S. Agarwal, "Comparative Study of Big Data Computing and Storage

Tools : A Review," *Int. J. Database Theory Appl.* , vol. 9, no. 1, pp. 45–66, 2016.

[34] S. Landset, T. M. Khoshgoftaar, A. N. Richter, and T. Hasanin, "A survey of open source tools for machine learning with big data in the Hadoop ecosystem," *J. Big Data*, vol. 2, no. 1, p. 24, 2015.

[35] V. B. Bobade, "Survey Paper on Big Data and Hadoop," *Int. Res. J. Eng. Technol.*, vol. 3, no. 1, pp. 861–863, 2016.

[36] A. Grover *et al.*, "SQL-like big data environments: Case study in clinical trial analytics," *Big Data (Big Data), 2015 IEEE Int. Conf.*, pp. 2680–2689, 2015.

[37] N. Mouthaan, "Effects of Big Data Analytics on Organizations ' Value Creation," *Nielsmouthaan.Nl*, no. 6377289, 2012.

[38] X. Liu, N. Iftikhar, and X. Xie, "Survey of real-time processing systems for big data," *Proc. 18th Int. Database Eng. Appl. Symp. - IDEAS '14*, pp. 356–361, 2014.

[39] Jethro, "Hadoop Hive and 11 SQL-on-Hadoop Alternatives," 2016. [Online]. Available: https://jethro.io/hadoop-hive.

[40] MapR, "SQL on Hadoop Details," 2017. [Online]. Available: https://mapr.com/why-hadoop/sql-hadoop/sql-hadoop-details/.

[41] M. Y. Santos *et al.*, "Evaluating SQL-on-Hadoop for Big Data Warehousing on Not-So-Good Hardware," *Proc. 21st Int. Database Eng. Appl. Symp. - IDEAS 2017*, pp. 242–252, 2017.

[42] C. McDonald, "Apache Drill Architecture: The Ultimate Guide," 2015. [Online]. Available: https://www.mapr.com/blog/apache-drill-architecture-ultimate-guide.

[43] MapR, "SQL on Hadoop Details," 2016. [Online]. Available: https://mapr.com/why-hadoop/sql-hadoop/sql-hadoop-details/.

[44] D. Lynn, "SQL ON HADOOP THE DIFFERENCES AND MAKING THE RIGHT CHOICE," 2016. [Online]. Available: http://www.agildata.com/sql-on-hadoop-the-differences-and-making-the-right-choice/.

[45] I. Szegedi, "Pivotal Hadoop Distribution and HAWQ Realtime Query Engine," 2014. [Online]. Available: https://dzone.com/articles/pivotal-hadoop-distribution.

[46] S. Sakr, "A brief comparative perspective on SQL access for Hadoop," in *Recent Advances in Information Systems and Technologies, Volume 1*, 2014, pp. 1–9.

[47] M. Mokhtar, "HIVE 0.14 COST BASED OPTIMIZER (CBO) TECHNICAL OVERVIEW," 2015. [Online]. Available: https://hortonworks.com/blog/hive-0-14-cost-based-optimizer-cbo-technical-overview/.

[48] A. Sebaa, F. Chikh, A. Nouicer, and A. Tari, "Research in Big Data Warehousing using Hadoop," *J. Inf. Syst. Eng. Manag.*, vol. 2, no. 2, pp. 1–5, 2017.

[49] Hortonworks, "Hive Performance Tuning," 2016.

[50] Y. N. Silva, I. Almeida, and M. Queiroz, "SQL: From Traditional Databases to Big Data," *Proc. SIGCSE - ACM Tech. Symp. Comput. Sci. Educ.*, p. 6, 2016.

[51] S. Bhuiyan, "Ad-hoc analysis over Cassandra data with Facebook Presto," 2014. [Online]. Available: http://frommyworkshop.blogspot.pt/2014/05/ad-hoc-analysis-

over-cassandra-data.html.

[52] J. Laskowski, *Mastering Apache Spark 2.0*. 2016.

[53] T. Sakaki, M. Okazaki, and Y. Matsuo, "Earthquake Shakes Twitter Users: Real-time Event Detection by Social Sensors," *Proc. 19th Int. Conf. World Wide Web*, pp. 851–860, 2010.

[54] A. Grover *et al.*, "SQL-like big data environments: Case study in clinical trial analytics," *2015 IEEE Int. Conf. Big Data (Big Data)*, pp. 2680–2689, 2015.

[55] A. Grover *et al.*, "SQL-like big data environments: Case study in clinical trial analytics," *2015 IEEE Int. Conf. Big Data (Big Data)*, pp. 2680–2689, 2015.

[56] T. P. Morgan, "EMC morphs Hadoop elephant into SQL database Hawq," 2013. [Online]. Available: http://www.theregister.co.uk/2013/02/25/emc_pivotal_hd_hadoop_hawq_database/.

[57] T. P. P. Council, "TPC BenchmarkTM H Standard Specification Revision 2.17.2," 2017.

[58] A. R. Reserved, "TPC BENCHMARK ™ DS Standard Specification Transaction Processing Performance Council (TPC)," 2014.

[59] C. Shanklyn, "ORCFILE IN HDP 2: BETTER COMPRESSION, BETTER PERFORMANCE," 2017. [Online]. Available: https://hortonworks.com/blog/orcfile-in-hdp-2-better-compression-better-performance/.

[60] B. D. Black, "SQL Engines for Hadoop: Hive vs Impala vs Spark," 2017. .

[61] J. Bernardino and P. Neves, "Decision-Making with Big Data Using Open Source Business Intelligence Systems.," in *Human Development and Interaction in the Age of Ubiquitous Technology*, IGI Global, 2016, pp. 120–147.

[62] B. R. Prasad and S. Agarwal, "Comparative Study of Big Data Computing and Storage Tools : A Review," *Int. J. Database Theory Appl.* , vol. 9, no. 1, pp. 45–66, 2016.

[63] S. Shinde, "Apache Hive Or Cloudera Impala? What is Best for me?," 2013. .

[64] Y. N. Silva, I. Almeida, and M. Queiroz, "SQL: From Traditional Databases to Big Data," *Proc. SIGCSE - ACM Tech. Symp. Comput. Sci. Educ.*, p. 6, 2016.

Appendix A – Worldcist Published Paper

Describing and Comparing Big Data Querying Tools

Mário Rodrigues[1], Maribel Yasmina Santos[2,] Jorge Bernardino[1,3]

[1] Polytechnic of Coimbra, Institute of Engineering of Coimbra (ISEC), Coimbra, Portugal
[2] ALGORITMI Research Centre, University of Minho, Guimarães, Portugal
[3] CISUC – Centre for Informatics and Systems of the University of Coimbra
a21190357@alunos.isec.pt, maribel@dsi.uminho.pt, jorge@isec.pt

Abstract. In the past years, Big Data has become a hot topic across several business areas. One of the main concerns regarding this concept is how to handle the massive volume and variety of data efficiently. Due to the notorious complexity of the data associated to the Big Data concept, usually motivated by data volume, efficient querying analysis mechanisms are mandatory for data analysis purposes. Motivated by the rapidly development of tools and frameworks for Big Data, there is much discussion about querying tools and, specifically, those more appropriated for specific analytical needs. This paper explores some of the available querying tools, describing and comparing their main characteristics and architectures, crucial knowledge for selecting the more appropriate ones for inclusion in a specific Big Data analytical architecture.

Keywords: Big Data, SQL-on-Hadoop, Query Processing, Analytics.

1 Introduction

As a consequence of the tremendous increase of the Internet usage, like social media, daily online operations and transactions from multiple sources, the amount of generated data has grown exponentially. We live in a world of data since everything we do in our lives is leaving a footprint or trace. In this context, organizations are starting to realize the importance of taking advantage of data to support the decision-making process.

Despite the complexity and the many challenges associated to the use of the Big Data concept, we cannot ignore the potential lying in it. It can support for analytics and for the identification of hidden patterns, which are very effective in defining business strategies. In the Big Data scope, Structured Query Language (SQL) processing has gained significant attention, as many enterprise data management tools rely on SQL, and, also, many users are familiar and comfortable with it [1]. For an organization, this opens up Big Data to a much larger audience, increasing the return of investment in this field [61].

Due to the extensive use of SQL, the number of SQL-on-Hadoop systems has increased significantly, allowing users to perform ad-hoc querying and analysis. However, not all SQL querying tools are equal, challenging the selection of the most appropriate one. In this paper, an overview of some Big Data querying tools is presented, describing and comparing their main characteristics.

The analyzed tools are Drill, HAWQ, Hive, Impala, Presto and Spark. The main contributions of this paper are:

- Provide an overview of some of the existing Big Data querying tools, describing their main characteristics;
- Highlight and define the essential features that a certain querying tool engine must have to efficiently process queries.

The rest of the paper is organized as follows. Section 2 presents the related work. Section 3 characterizes the compared tools and their architectures. Section 4 presents a comparative table for the selected tools. Finally, section 5 summarizes the presented work and presents some proposals for future work.

2 Related Work

This phenomenon called Big Data, identified as one of the biggest IT trends of the last few years, includes a large amount of work regarding Big Data querying and processing. With this growth, SQL processing, namely SQL-on-Hadoop, has been widely studied, analysing and evaluating the performance of several processing tools.

In this context, the work of [1] provides a performance comparison of Hive and Impala using the TPC-H benchmark and two TPC-DS inspired workloads, analysing the I/O efficiency of their columnar formats. The results show that Impala is faster than Hive, on either MapReduce and on Tez for the overall TPC-H and TPC-DS benchmarks. The work performed on [46] presents the reasons why the author thinks we should use SQL access on Hadoop, also giving an overview of IBM Big SQL and introducing Hive, Impala and HAWQ. In [32], the authors present Impala giving an overview of its architecture and main components, also demonstrating its performance when compared against other popular SQL-on-Hadoop systems like Spark, Presto and Hive, revealing that Impala has faster response times executing queries in single and multi-user query execution. In [23], recent trends of storage and computing tools are analysed, showing their relative capabilities, limitations and environment they are suitable to work with, and [62] presents a detailed description of four storage tools (HBase, Hive, Neo4j and Cassandra) and four computing tools (Hadoop MapReduce, Impala, IBM Netezza and Giraph). In [34], the integration of Machine Learning in the Hadoop ecosystem is analysed, studying three different processing paradigms along with a comparison of engines that implement them, like MapReduce, Spark, Flink or Storm. Also, the paper presents a comparison of machine learning libraries, including Mahout, MLlib and SAMOA based on scalability, ease of use and extensibility. In [35], a study on HDFS, MapReduce, Pig, Hive, HBase and Spark is performed, describing their main characteristics and architectures. Another benchmark of SQL-like Big Data technologies is presented in [36], applying them in clinical trial databases, comparing Hive, Presto, Drill and Spark. In [39], Hive proprietary and alterative Open Source tools are presented, analysing each one against the needs of Big Data processing. In [9, 10], the motivation for using Hadoop is presented, along with the strengths and limitations of tools like Impala, Hive, BigSQL, HAWQ and Presto. On [43], characteristics like latency and ANSI SQL completeness, are used to evaluate Drill, Hive, Impala and SparkSQL. The work of [30] uses TPC-DS queries, comparing response times for a single user and for 10 concurrent users, using Impala, Hive and Spark.

Although a comprehensive set of works is available, most of the related work concentrates on practical approaches and benchmarks, comparing query execution times and other performance aspects, not specifying scenarios for selecting the tools. This work provides an overview of a set of selected Big Data querying tools through the analysis of the state of the art, describing and comparing their characteristics and architectures in order to conclude in what scenario each tool should be used.

3 BIG DATA QUERYING TOOLS

The development and use of highly distributed and scalable systems to process Big Data is considered as one of the recent key technological developments. Some of these systems have proposed proprietary query languages or application program interfaces, while others have recognized the benefits of using SQL. Not all SQL querying tools are equal, and that makes a challenge of picking the right tool. For comparison and due to their popularity, this work analyses and compares Drill, HAWQ, Hive, Impala, Presto and Spark.

3.1 Drill

Drill is an open-source distributed system that supports data-intensive distributed applications for interactive analysis of large-scale datasets. It was developed with the goal of providing low latency and faster interactive queries; it supports several data storage NoSQL databases like HDFS, Hive or HBase and SQL to query data, being capable of performing mathematical and statistical functions, string and dates manipulation. Besides the mentioned data storages, data can reside in different file formats like CSV, TSV, JSON, PARQUET and AVRO. Like Hive, it allows creating custom functions in JAVA code. With the simple installation, it can scale-up to a very large cluster with thousands of nodes. On top of all these characteristics, it offers connectivity and compatibility with BI tools like Tableau, Microstrategy, Qlikview and Tibco. Its core is the 'Drillbit' service, which is responsible for accepting requests from the client, processing the queries, and returning results to the client. Drillbit can be installed and run on all of the required nodes in a Hadoop cluster, forming a distributed cluster environment, using ZooKeeper to maintain cluster membership, health-check information and identifying the appropriate nodes to execute query fragments.

The Drillbits are composed by components like: i) RPC end-point, allowing communication with the clients and receiving queries through the RPC protocol; ii) SQLParser, optimizing queries and generating a distributed query plan that is optimized for fast and efficient execution; and, iii) Optimizer, responsible for managing standard database optimizations, providing a distributed query plan for efficiently queries execution across different nodes. In overall, Apache Drill provides interactive query capabilities enabling traditional BI and analytics from different sources, suited for advanced analytic workflows, offering response times that vary between milliseconds to minutes depending on the query complexity [43]. Drill provides a unified query layer that can interact with different file formats in different data sources avoiding any ETL, which makes it appropriate in any environment that need to query data from several data sources.

3.2 HAWQ

HAWQ (Hadoop With Query) is a Hadoop native SQL query engine that combines the key technological advantages of MPP with the scalability and convenience of Hadoop, offering good performance. Developed by Pivotal, it adopts a layered architecture and relies on HDFS for data replication and fault tolerance. Also, HAWQ relies on both the PostgreSQL (or Greenplum) database and the HDFS storage as backend storage mechanism, meaning that HAWQ can support the full SQL syntax, being a good choice when we need interactive real-time analytics and short response times. To perform complex queries, it breaks them into small tasks and distributes them to MPP query processing units for execution. Applications and external tools can interact with HAWQ via standard protocols, such as JDBC, ODBC, and libpq, used by PostgreSQL. The architecture of the HAWQ engine includes the following main components [46]: i) *HAWQ master*: entry point responsible for accepting the connections from the clients and manages the system tables that contain metadata information about HAWQ itself, being also responsible for parsing and optimizing the queries and generating the query execution plan; ii) *HAWQ segments*: represent the processing units, responsible for running the local database operations on their own data sets; iii) *HAWQ storage nodes*: storing all the user data (HAWQ relies on a proprietary file format for storing the HDFS data); and, iv) *HAWQ interconnect*: responsible for managing the inter-process communication between the segments during query execution. Another capability of HAWQ is its integration with MADlib, providing machine-learning capabilities directly in SQL [60].

3.3 Hive

Hive is a system that supports the processing and analysis of data stored in Hadoop. It is considered to be the first to support SQL-on-Hadoop and has been used by many organizations such as Amazon. Built on the Hadoop platform, it supports familiar relational database concepts such as tables, columns, and partitions and includes some SQL support for unstructured data [1]. Hive gives structure onto data and perform *ad-hoc* queries and analysis using HiveQL, a SQL-like query language. Hive works by storing the metadata in Hive Metastore, used to provide a schema on read functionality and semantics check on queries [28] Using the information available in the Metastore, the Hive server transforms HiveQL queries into MapReduce jobs, enabling users to focus on specifying queries and how they will perform.

Although HiveQL is a SQL-like querying language, there are some SQL features not yet available like the lack of support for date or time variables (they are treated as strings [46], for UPDATE or DELETE, and INSERT operations in single rows.

In terms of processing, Hive executes queries and stores them in the HDFS. As HiveQL does not feature the full capacity of SQL, it allows to plug-in custom user defined functions (UDFs) and aggregation functions (UDAFs) written in Java. In terms of processing, SQL queries are submitted to Hive and executed as follows: i) Hive compiles the query; ii) an execution engine, like Tez or MapReduce, executes the compiled query; iii) the resource manager, YARN, allocates resources for applications across the Hadoop cluster; iv) the data manipulated in the query is maintained in HDFS; and, v) query results are made available over a JDBC/ODBC connection.

In order to improve query performance, Hive creates indexes to accelerate queries and as query editor, users can use Hue, a web-based application that can be used to write and run HiveQL queries. In general, Hive has a good interface for anyone from the relational database world, demanding low maintenance and being simple to learn. We can think in Hive as a standard, since all Hadoop distributions include it. Originally, Hive relied on Hadoop's MapReduce suffering from poor performance, being more appropriate for large scans, where query performance may not be so critical, and not for OLTP (Online Transactions Processing). It is worth mentioning that using MapReduce is the main criticisms made to Hive, as the conversion to MapReduce jobs leads to higher query latency. Later versions of Hive can run on Tez, a tool that aims to enhance performance, delivering Hive interactive resulting in improved query performance and latency. Moreover, and as Hive is not intended for OLTP, users cannot expect operations for row level updates.

3.4 Impala

Impala is an open-source, state-of-the-Art Massive Parallel Processing (MPP) SQL query engine designed for performance, real time, low latency and high concurrency processing. It was developed in C++ and Java, with the objective of combining the SQL support and multi-user performance of a traditional analytic database with the scalability and flexibility of Apache Hadoop. Also, it provides a C++ API for user defined functions (UDFs). Impala implements a distributed architecture that can run on hundreds of machines in existing Hadoop clusters and has been built to extend the key components of Hive, e.g., SQL syntax (HiveQL), metadata and schema [50]. Impala is part of the Hadoop ecosystem and shares some of its infrastructures, as metadata, Hive, and Pig.

Impala can use two storage systems, HDFS or HBase, being possible to query data in both. SQL queries are performed on top of HBase tables, which runs on top of HDFS and provides a fault-tolerant way to store and process large amounts of sparse tabular data. It can also integrate with Business Intelligence (BI) tools like Tableau, Pentaho, Micro Strategy and Zoom Data, being a good choice in environments where there is the need of interactive real-time analytics, e.g., retail, where users need fast methods that provide deep insights into price sensitivities and behavior at the customer level over a period of time. One of the main limitations of Impala is that it relies on in-memory join implementations, meaning that queries can fail if the joined tables cannot fit into memory [46] Other drawbacks are: i) the lack of support for serialization and deserialization and, whenever records or files are added to the data directory in HDFS, the tables need to be refreshed; ii) the lack of support on fault tolerance. If a node fails in the middle of processing, the whole query has to be restarted. In summary, Impala gains big advantage in real-time and *ad-hoc* queries, as long as the runtime is short enough that node failures during the query execution are unlikely [63].

3.5 Presto

Presto is an open source distributed SQL query engine for running interactive analytic queries. It was developed by Facebook to give Hadoop some Data Warehouse and SQL-like capabilities, being optimized for low latency and interactive query analysis. Facebook uses Presto for interactive queries against several internal data stores, including its 300PB Data Warehouse. Over 1,000 Facebook employees use Presto daily to run more than 30,000 queries, each scanning over a petabyte per day. Presto supports several features of SQL including joins, aggregations and subqueries, as well as other features that include JSON, URL functions, strings and regular expression functions. A single Presto query can combine data from multiple sources, allowing for analytics across several data sources in the organization, meaning that it was not only designed for querying data present in HDFS, but also in other data sources, including relational or non-relational databases. It uses in-memory processing, instead of MapReduce, in order to avoid sidesteps, unnecessary I/O and associated latency. As a result, Facebook claims that Presto runs 10 times faster than Hive [36]. Presto is suitable for interactive analysis, providing quick answers, being capable of aggregate large amounts of data and produce reports.

Presto can perform simple queries in few hundred milliseconds and more complex in few minutes. It uses memory much more aggressively than Hive, keeping the intermediate data in memory, rather than using disk. However, Hive is still necessary for some operations, like loading data. Running on a cluster of machines, it operates having as base an architecture that includes a coordinator, multiple workers and a client: i) *Presto CLI*, a terminal-based interactive shell that submits SQL statements to a coordinator to get the result; ii) *Presto Coordinator*, parsing the SQL queries for defining the query execution plan; iii) *Presto Workers*, receiving assignments from the coordinator and delivering results. One of the main drawbacks of Presto is the limitation on the maximum amount of memory each query can have, meaning that a query will fail if it requires a large amount of memory.

3.6 Spark

Spark is a highly distributed processing framework that provides an ease of use tool for efficient analytics on heterogeneous data. Its main characteristic is the way it works by loading the data into a cluster's memory and performing the necessary queries. Spark's performance has been reported to be up to one hundred times faster than Hadoop's MapReduce [64]. Spark uses the concept of Resilient Distributed Datasets (RDD), which describes an immutable collection of objects that are partitioned and distributed across multiple nodes of a cluster, allowing parallel processing. One of the key modules of Spark is Spark SQL, used for processing structured data with DataFrames (equivalent to relational database tables) that organize data in named columns. In order to extend the vocabulary of SparkSQL, it is possible to plug-in custom user defined functions (UDFs). Spark uses a master/worker architecture, where a single coordinator called master manages all workers, executing tasks. Spark is designed for supporting a large range of workloads like batch applications, iterative algorithms, and interactive queries and streaming, through the following components: i) Spark Streaming, leveraging fast scheduling capability to perform streaming analytics with Spark Core; ii) MLib (machine

learning), a distributed machine learning framework above Spark, which is nine times faster than Hadoop disk-based version of Apache Mahout [35]; and, iii) GraphX, a distributed graph-processing framework on top of Spark for simplifying analytical tasks. These components allow real-time analysis, discovery and processing patterns, complex math, statistics, or machine learning, being a good choice in real time processing scenarios, e.g. event detection. In [53], the authors describe how we could detect an earthquake by analyzing Twitter real-time data, showing that this technique is able to inform about an earthquake in Japan quicker than the Japan Meteorological agency. One of the main disadvantages of Spark is the high consumption of memory.

4 Comparing Big Data Processing Tools

As previously mentioned, Big Data is frequently unstructured, with different formats and requiring the integration of several sources, so efficient querying processing structure is demanded. Based on the works of [34], [36] and [43] the following characteristics can be defined as the main requirements for an efficient Big Data processing architecture: i) *Scalability*: Scaling is the ability of the system to adapt to increased demands in terms of data processing; linear scalability is necessary for the explosive growth of data size; ii) *High throughput* (processing speed): Big Data's velocity demands that data be ingested and processed at high speeds, requiring an infrastructure that is extremely fast across input/output (I/O), processing, and storage; iii) *High degree of parallelism*: By processing data in parallel, we can distribute the load across multiple machines, each one having its own copy of the data, but processing a different part; iv) *Programming language support*: The support of programming languages can be useful for integrating different BI tools or for supporting other developments; v) *SQL Support*: SQL is the base for queries processing; vi) *Distributed Architecture*: Distributed processing across several servers (nodes), providing parallel computing; vii) *Fault Tolerance*: Mechanisms for detecting failures and for recovering from them; viii) *Single Point of Failure*: Identifying when the whole query is aborted if one of the processing nodes fails; and, ix) *Machine Learning Algorithms/Tools*: For advanced data processing, allowing the identification of hidden patterns and trends in Big Data.

Considering the querying engines and their core features presented in the previous sections, Table 1 compares Drill, HAWQ, Hive, Impala, Presto and Spark attending to the identified characteristics.

Table 20. Comparative table of the Big Data Processing Tools.

Feature	**Drill**	**HAWQ**	**Hive**	**Impala**	**Presto**	**Spark**
Owner	Community	Greenplum	Community	Cloudera	Facebook	Community
Cluster Size Limit (nodes)	Thousands	Thousands	Hundreds	Thousands	Thousands	Thousands
SQL support	ANSI SQL	ANSI SQL	HiveQL	HiveQL	Not fully ANSI SQL compliant (subqueries)	ANSI SQL (limited) & HiveQL
Latency	Low	Low	Medium	Low	Low	Low
Scalability	High	High	High	Very High	Very High	High
Machine Learning	No	Yes	No	Yes	No	Yes
Data Visualization	No Native support, compatibility with BI tools	No Native support, compatibility with BI tools	No Native support, compatibility with BI tools	No Native support, compatibility with BI tools	No Native support, compatibility with BI tools	No Native support, compatibility with BI tools
Processing Speed	Fast	Fast	Slow	Very Fast	Very Fast	Fast
Language Support	C++, Java and other languages JDBC/ ODBC	Python, Perl, Java, C/C++, R	Java	All languages supporting JDBC/ ODBC	C, Java, Node.js, PHP, Python, R, Ruby	Java, Python, R, Scala, JDBC/ ODBC
Use Cases	Real-time Interactive analysis/BI (Ad-hoc queries)	Batch processing	Batch processing	Real-time Interactive analysis/BI (Ad-hoc queries)	Real-time Interactive analysis	Batch and streaming, interactive queries and ML
Distributed Architecture	Yes	Yes	Yes	Yes	Yes	Yes
Single Point of Failure in	No	No	Yes, if failure at	Yes, if any host quits	Yes, if any host quits	No

Feature	Drill	HAWQ	Hive	Impala	Presto	Spark
Query Execution			master node.	query execution	query execution	
Fault Tolerance	Yes	Yes	Yes	No	No	Yes
File/Storage Formats	CSV, TSV, PSV, Parquet, Hadoop Sequence Files (Key-value pairs)	Text, Parquet, Avro, CSV	ORC, AVRO, Parquet, and Text	Parquet, Text, Avro, RCFile or Sequence files	Text, Sequence Files, RCFile, ORC and Parquet	JSON, Parquet, CSV, ORC, RDDs, Hive Tables and External Databases

Comparing Impala and Hive, both support SQL but Hive is more appropriate for long-running batch processing, offering robustness and low maintenance. Impala provides a better support for real-time querying having less latency than Hive, since it bypasses MapReduce. Using Hive with Tez, instead of MapReduce, makes Hive a possible solution for real-time analytics/ad hoc querying and OLTP environments, but based on some benchmarks (e.g [30], [32]), Impala can provide better response times.

In terms of fault tolerance, Hive easily retries queries in case of failure, which does not happen in Impala. Spark has a machine learning library (MLib), which makes it suitable for analytics, being an adequate choice when the objective is not just querying data but working with it in an exploratory manner, even for real-time data streaming. Spark has short query response times, being faster than Hive, even so, Impala seems to be faster [32] [30]. Drill suits best when the main objective is only querying data from several data sources. It has better response times than Spark, which does not happen in case of a single source. It is also fully ANSI SQL compliant. Another suited solution for combining data from multiple sources is Facebook Presto, performing queries with concerns with CPU efficiency and latency, having faster response times than Drill. In some works Cloudera claims that Impala is 5.3 to 7.5 times faster than Presto [10].

Of all the presented tools, after the analysis of the state of the art, HAWQ is the less known option. However, it can provide strong processing with full ANSI SQL support and provides machine learning and data mining algorithms. On several benchmarks, HAWQ managed to achieve performance improvements of 10x to 600x [56], surpassing Impala, improvements that could turn batch systems into interactive ones.

5 Conclusions and Future Work

Looking at the characteristics and architectures of the big data querying tools discussed above, we consider that there is no one-fits-all tool. We conclude that Hive is a standard in the field of Big Data, since it is considered to be the first to support SQL-on-Hadoop. However, it is for batch processing systems, being chosen due to its robustness for long-term queries. In terms of real-time interactive analysis, which requires very fast response times, Impala and HAWQ seem to be the best tools. Nevertheless, and considering that Impala has great performance and is very popular, HAWQ seems to surpass it in query response times, with the advantage of being fully ANSI SQL compliant (Impala only supports HiveQL). When the objective is not only querying data, but performing advanced analyses, statistics and Machine Learning, Spark can be a good option, as well as Impala and HAWQ, although Spark is the only tool suited for stream processing, making it the choice when dealing with continuous data streams.

As future work we propose to use standard benchmarks to perform an experimental evaluation of the analyzed Big Data querying tools, getting a practical view of the performance and suitability for specific uses.

Acknowledgments. This work has been supported by COMPETE: POCI-01-0145-FEDER-007043 and FCT (*Fundação para a Ciência e Tecnologia*) within the Project Scope: UID/CEC/00319/2013, and by Portugal Incentive System for Research and Technological Development, Project in co-promotion nº 002814/2015 (iFACTORY 2015-2018).

References

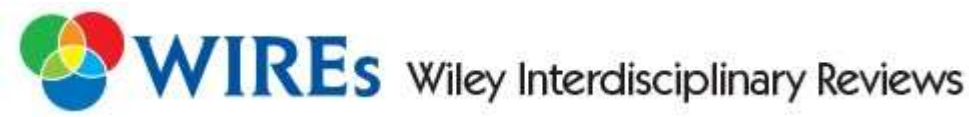
[1] A. Floratou, U. F. Minhas, and F. Ozcan, "Sql-on-hadoop: Full circle back to shared-nothing database architectures," *Proc. VLDB Endow.*, vol. 7, no. 12, pp. 1295–1306, 2014.

[2] E. G. Ularu, F. C. Puican, A. Apostu, and M. Velicanu, "Perspectives on Big Data and Big Data Analytics," *Database Syst. J.*, vol. III, no. 4, pp. 3–14, 2012.

[3] A. Gandomi and M. Haider, “Beyond the hype : Big data concepts , methods , and analytics,” *Int. J. Inf. Manage.*, vol. 35, pp. 137–144, 2015.
[4] C. Costa, “BASIS : Uma Arquitetura de Big Data para Smart Cities,” Universidade do Minho, 2015.
[5] M. Chen, S. Mao, and Y. Liu, “Big data: A survey,” *Mob. Networks Appl.*, vol. 19, no. 2, pp. 171–209, 2014.
[6] George Firican, “The 10 Vs of Big Data,” 2017. [Online]. Available: https://tdwi.org/articles/2017/02/08/10-vs-of-big-data.aspx.
[7] A. DeVan, “The 7 V’s of Big Data,” 2016. [Online]. Available: https://www.impactradius.com/blog/7-vs-big-data/.
[8] R. Maheshwari, “3 V’s or 7 V’s - What’s the Value of Big Data?,” 2015. [Online]. Available: https://www.linkedin.com/pulse/3-vs-7-whats-value-big-data-rajiv-maheshwari/.
[9] B. Y. Mike, “Big Data and Business Intelligence : a data-driven strategy for e-commerce organizations in the hotel industry Big Data and Business Intelligence : a data-driven strategy for e-commerce organizations in the hotel industry,” p. 57, 2015.
[10] C. Costa and M. Y. Santos, “Big Data : State-of-the-art Concepts , Techniques , Technologies , Modeling Approaches and Research Challenges,” *IAENG Int. J. Comput. Sci.*, vol. 44, no. 3, pp. 285–301, 2017.
[11] R. Pandya, V. Sawant, N. Mendjoge, and M. D ’silva 4, “Big Data Vs Traditional Data,” vol. 3, no. X, pp. 192–196, 2015.
[12] P. N. Amundrud, “Opportunities for Automation, Internet of Things, Big Data Analytics and 3D Printing within Oil and Gas Drilling, Production and Transport.,” no. June, 2009.
[13] A. C. Gessler, “MapReduce to Couple a Bio-mechanical and a Systems-biological Simulation,” no. 156, 2014.
[14] S. Mehta and V. Mehta, “Hadoop Ecosystem : An Introduction,” vol. 5, no. 6, pp. 557–562, 2016.
[15] B. PROFFITT, “Hadoop: What It Is And How It Works,” 2013. [Online]. Available: https://readwrite.com/2013/05/23/hadoop-what-it-is-and-how-it-works/.
[16] TutorialsPoint, “Hadoop - MapReduce,” 2017. [Online]. Available: https://www.tutorialspoint.com/hadoop/hadoop_mapreduce.htm.
[17] S. Ahmed, “Evaluating Presto as an SQL on Hadoop Solution Evaluating Presto as an SQL on Hadoop solution: A Case at Truecaller,” 2016.
[18] K. Krishnan, *Data Warehousing in the Age of Big Data*. 2013.
[19] A. Oussous, F. Z. Benjelloun, A. Ait Lahcen, and S. Belfkih, “Big Data technologies: A survey,” *J. King Saud Univ. - Comput. Inf. Sci.*, 2017.
[20] “Cloudera vs. Hortonworks vs. MapR - Hadoop Distribution Comparison,” 2016. [Online]. Available: https://www.dezyre.com/article/cloudera-vs-hortonworks-vs-mapr-hadoop-distribution-comparison-/190.
[21] Cloudera, “CDH: The Most Popular Open Source Distribution of Apache Hadoop,” 2017. [Online]. Available: cdh: The Most Popular Open Source Distribution of Apache Hadoop.
[22] “Benchmarking Big Data SQL Frame- works,” 2016.
[23] J. Bernardino and V. Abramova, “No Experimental Evaluation of NoSQL Databases,” *Int. J. Database Manag. Syst.*, vol. 6, pp. 1–16, 2014.
[24] R. J. de S. Marques, “Big data model with SQL support for performance management in telecommunication networks,” University of Aveiro, 2014.
[25] T. T. Maposa and M. Sethi, “SQL-on-Hadoop : The Most Probable Future in Big Data Analytics,” *Adv. Comput. Sci. Inf. Technol.*, vol. 2, no. 13, pp. 9–14, 2015.
[26] B. Saha, H. Shah, S. Seth, G. Vijayaraghavan, A. Murthy, and C. Curino, “Apache Tez,” *Proc. 2015 ACM SIGMOD Int. Conf. Manag. Data - SIGMOD ’15*, pp. 1357–1369, 2015.
[27] S. Van Wouw, J. Viña, A. Iosup, and D. Epema, “An Empirical Performance Evaluation of Distributed SQL Query Engines: Extended Report,” 2014.
[28] C. Owl, “The SQL on Hadoop landscape: An overview (Part I),” 2015. [Online]. Available: http://cleverowl.uk/2015/11/19/the-sql-on-hadoop-landscape-an-overview-part-i/.
[29] C. Owl, “The SQL on Hadoop landscape: An overview (Part II),” 2015. [Online]. Available: http://cleverowl.uk/2015/12/25/the-sql-on-hadoop-landscape-an-overview-part-ii/.
[30] D. K. Devadutta Ghat, David Rorke, “New SQL Benchmarks: Apache Impala (incubating) Uniquely Delivers Analytic Database Performance,” 2016. [Online]. Available: https://blog.cloudera.com/blog/2016/02/new-sql-benchmarks-apache-impala-incubating-2-3-uniquely-delivers-analytic-database-performance/.
[31] S. Sakr, “A brief comparative perspective on SQL access for Hadoop,” in *Recent Advances in Information Systems and Technologies, Volume 1*, 2014, pp. 1–9.
[32] M. Kornacker *et al.*, “Impala: A Modern, Open-Source SQL Engine for Hadoop,” *CIDR(Conference Innov. Data Syst. Res.)*, 2015.
[33] B. R. Prasad and S. Agarwal, “Comparative Study of Big Data Computing and Storage Tools : A Review,” *Int. J. Database Theory Appl.* , vol. 9, no. 1, pp. 45–66, 2016.
[34] S. Landset, T. M. Khoshgoftaar, A. N. Richter, and T. Hasanin, “A survey of open source tools for machine learning with big data in the Hadoop ecosystem,” *J. Big Data*, vol. 2, no. 1, p. 24, 2015.
[35] V. B. Bobade, “Survey Paper on Big Data and Hadoop,” *Int. Res. J. Eng. Technol.*, vol. 3, no. 1, pp. 861–863, 2016.
[36] A. Grover *et al.*, “SQL-like big data environments: Case study in clinical trial analytics,” *Big Data (Big Data), 2015 IEEE Int. Conf.*, pp. 2680–2689, 2015.
[37] N. Mouthaan, “Effects of Big Data Analytics on Organizations ’ Value Creation,” *Nielsmouthaan.Nl*, no. 6377289, 2012.
[38] X. Liu, N. Iftikhar, and X. Xie, “Survey of real-time processing systems for big data,” *Proc. 18th Int. Database Eng. Appl. Symp. - IDEAS ’14*, pp. 356–361, 2014.
[39] Jethro, “Hadoop Hive and 11 SQL-on-Hadoop Alternatives,” 2016. [Online]. Available: https://jethro.io/hadoop-

hive.
[40] MapR, “SQL on Hadoop Details,” 2017. [Online]. Available: https://mapr.com/why-hadoop/sql-hadoop/sql-hadoop-details/.
[41] M. Y. Santos *et al.*, “Evaluating SQL-on-Hadoop for Big Data Warehousing on Not-So-Good Hardware,” *Proc. 21st Int. Database Eng. Appl. Symp. - IDEAS 2017*, pp. 242–252, 2017.
[42] C. McDonald, “Apache Drill Architecture: The Ultimate Guide,” 2015. [Online]. Available: https://www.mapr.com/blog/apache-drill-architecture-ultimate-guide.
[43] MapR, “SQL on Hadoop Details,” 2016. [Online]. Available: https://mapr.com/why-hadoop/sql-hadoop/sql-hadoop-details/.
[44] D. Lynn, “SQL ON HADOOP THE DIFFERENCES AND MAKING THE RIGHT CHOICE,” 2016. [Online]. Available: http://www.agildata.com/sql-on-hadoop-the-differences-and-making-the-right-choice/.
[45] I. Szegedi, “Pivotal Hadoop Distribution and HAWQ Realtime Query Engine,” 2014. [Online]. Available: https://dzone.com/articles/pivotal-hadoop-distribution.
[46] S. Sakr, “A brief comparative perspective on SQL access for Hadoop,” in *Recent Advances in Information Systems and Technologies, Volume 1*, 2014, pp. 1–9.
[47] M. Mokhtar, “HIVE 0.14 COST BASED OPTIMIZER (CBO) TECHNICAL OVERVIEW,” 2015. [Online]. Available: https://hortonworks.com/blog/hive-0-14-cost-based-optimizer-cbo-technical-overview/.
[48] A. Sebaa, F. Chikh, A. Nouicer, and A. Tari, “Research in Big Data Warehousing using Hadoop,” *J. Inf. Syst. Eng. Manag.*, vol. 2, no. 2, pp. 1–5, 2017.
[49] Hortonworks, “Hive Performance Tuning,” 2016.
[50] Y. N. Silva, I. Almeida, and M. Queiroz, “SQL: From Traditional Databases to Big Data,” *Proc. SIGCSE - ACM Tech. Symp. Comput. Sci. Educ.*, p. 6, 2016.
[51] S. Bhuiyan, “Ad-hoc analysis over Cassandra data with Facebook Presto,” 2014. [Online]. Available: http://frommyworkshop.blogspot.pt/2014/05/ad-hoc-analysis-over-cassandra-data.html.
[52] J. Laskowski, *Mastering Apache Spark 2.0*. 2016.
[53] T. Sakaki, M. Okazaki, and Y. Matsuo, “Earthquake Shakes Twitter Users: Real-time Event Detection by Social Sensors,” *Proc. 19th Int. Conf. World Wide Web*, pp. 851–860, 2010.
[54] A. Grover *et al.*, “SQL-like big data environments: Case study in clinical trial analytics,” *2015 IEEE Int. Conf. Big Data (Big Data)*, pp. 2680–2689, 2015.
[55] A. Grover *et al.*, “SQL-like big data environments: Case study in clinical trial analytics,” *2015 IEEE Int. Conf. Big Data (Big Data)*, pp. 2680–2689, 2015.
[56] T. P. Morgan, “EMC morphs Hadoop elephant into SQL database Hawq,” 2013. [Online]. Available: http://www.theregister.co.uk/2013/02/25/emc_pivotal_hd_hadoop_hawq_database/.
[57] T. P. P. Council, “TPC BenchmarkTM H Standard Specification Revision 2.17.2,” 2017.
[58] A. R. Reserved, “TPC BENCHMARK ^{TM} DS Standard Specification Transaction Processing Performance Council (TPC),” 2014.
[59] C. Shanklyn, “ORCFILE IN HDP 2: BETTER COMPRESSION, BETTER PERFORMANCE,” 2017. [Online]. Available: https://hortonworks.com/blog/orcfile-in-hdp-2-better-compression-better-performance/.
[60] B. D. Black, “SQL Engines for Hadoop: Hive vs Impala vs Spark,” 2017. .
[61] J. Bernardino and P. Neves, “Decision-Making with Big Data Using Open Source Business Intelligence Systems.,” in *Human Development and Interaction in the Age of Ubiquitous Technology*, IGI Global, 2016, pp. 120–147.
[62] B. R. Prasad and S. Agarwal, “Comparative Study of Big Data Computing and Storage Tools : A Review,” *Int. J. Database Theory Appl.* , vol. 9, no. 1, pp. 45–66, 2016.
[63] S. Shinde, “Apache Hive Or Cloudera Impala? What is Best for me?,” 2013. .
[64] Y. N. Silva, I. Almeida, and M. Queiroz, “SQL: From Traditional Databases to Big Data,” *Proc. SIGCSE - ACM Tech. Symp. Comput. Sci. Educ.*, p. 6, 2016.

Appendix B – WIREs Data Mining and Knowledge Discovery Submitted Paper



Article Title: Big Data Analytical Tools: An Experimental Performance Evaluation

Authors:

Mário Lucas Rodrigues [ORCID: 0000-0003-1065-414X]
Polytechnic of Coimbra, Institute of Engineering of Coimbra (ISEC), Coimbra, Portugal
a21190357@alunos.isec.pt

Jorge Rodrigues Bernardino [ORCID: 0000-0001-9660-2011]
Polytechnic of Coimbra, Institute of Engineering of Coimbra (ISEC), Coimbra, Portugal
CISUC – Centre of Informatics of University of Coimbra, Coimbra, Portugal jorge@isec.pt

Maribel Yasmina Santos [ORCID: 0000-0002-3249-6229]
ALGORITMI Research Centre, University of Minho, Guimarães, Portugal
maribel@dsi.uminho.pt

Abstract

In the past years, Big Data has become a hot topic across several business areas. One of the main challenges regarding this concept is how to handle the massive volume and variety of data efficiently. Due to the notorious complexity of the data associated to the Big Data concept, usually motivated by data volume, efficient querying analysis mechanisms are mandatory for data analysis purposes. Motivated by the rapidly development of tools and frameworks for Big Data, there is much discussion about querying tools and, specifically, those more appropriated for specific analytical needs. This paper describes and compares the main characteristics and architectures of the following popular Big Data analytical tools: Drill, Hive, Impala, Presto, and Spark. To test the performance of these Big Data analytical tools is also used TPC-H benchmark.

Keywords: Big Data, SQL-on-Hadoop, Query Processing, Big Data Analytics.

Introduction

In recent years the amount of generated data has grown exponentially as a consequence of the tremendous increase of the Internet usage, like social media, daily online operations and transactions from multiple sources. In this context, organizations are starting to realize the importance of taking advantage of data to support the decision-making process.

Despite the complexity and the many challenges associated to the use of the Big Data concept, we cannot ignore the potential lying in it. It can support for analytics and for the identification of hidden patterns, which are very effective in defining business strategies. In the Big Data scope, Structured Query Language (SQL) processing has gained significant attention, as many enterprise data management tools rely on SQL, and, also, many users are familiar and comfortable with it. For an organization, this opens up Big Data to a much larger audience, increasing the return of investment in this field (Floratou, Minhas, & Ozcan, 2014). Due to the extensive use of SQL, the number of SQLon-Hadoop systems has increased significantly, allowing users to perform ad-hoc querying and analysis. However, not all SQL querying tools have the same functionalities, being a challenge the selection of the most appropriate one.

In this paper we extended our previous work published in (Rodrigues, Santos, & Bernardino, 2017). The original paper describes the architecture and characteristics evaluate the following big data analytical tools: Drill, HAWQ, Hive, Impala, Presto, and Spark. This extended version, adds an experimental performance evaluation using the TPC-H benchmark. The main contributions of this paper are:

- A survey of most popular Big Data analytical tools, describing their main characteristics;
- Features and requirements comparison of Big Data analytical tools;
- Experimental evaluation of Big Data analytical tools using TPC-H benchmark.

The remainder of this article is organized as follows. First, we present the related work. In the following section we briefly described the compared tools. In the next sections we present a comparative table for the selected tools and perform a performance evaluation of analyzed tools. Finally, we conclude the paper and present some guidelines for future work.

Related Work

Big Data is identified as one of the biggest IT trends of the last few years, which includes a large amount of work regarding querying and processing tools. With this growth, SQL processing, namely SQL-on-Hadoop, has been widely studied, analyzing and evaluating the performance and of several processing tools.

In this context, the work performed on (Floratou et al., 2014) provides a performance comparison of Hive and Impala using the TPC-H benchmark and two TPC-DS inspired workloads, analyzing the I/O efficiency of their columnar formats. The results show that Impala is faster than Hive, on either MapReduce and on Tez for the overall TPC-H and TPC-DS benchmarks.

In (Owl, 2015a, 2015b) the motivation for using Hadoop is presented, along with the strengths and limitations of tools like Impala Hive, BigSQL, HAWQ, and Presto.

The work of (Sakr, 2014) presents a good overview of the reasons why we should use SQL access on Hadoop, also giving an overview of IBM Big SQL and comparing it with Hive, Impala, and Hawq. This work provides a good insight on SQL-on-Hadoop tools and shows that mainly Hawq and Impala can provide good performance.

In (Prasad & Agarwal, 2016), recent trends of storage and computing tools are analyzed, showing their relative capabilities, limitations and environment, they are suitable to work with. It is

presented a detailed description of four storage tools (HBase, Hive, Neo4j, and Cassandra) and four computing tools (Hadoop MapReduce, Impala, IBM Netezza, and Giraph). The results show that Cloudera Impala, IBM Netezza, and Apache Giraph can achieve very low latency time due to in-memory processing and Hive still does not provide OLTP.

In (Landset, Khoshgoftaar, Richter, & Hasanin, 2015) is studied the integration of Machine Learning in Hadoop ecosystem, studying three different processing paradigms (batch, iterative batch, and real-time streaming) along with a comparison of engines that implement them, like MapReduce, Spark, Flink or Storm. Also, the paper presents a comparison of machine libraries, including Mahout, MLlib and SAMOA.

In (Bobade, 2016) a study on HDFS, MapReduce, Pig, Hive, HBase, and Spark is performed, describing their main characteristics and architectures, showing the cost-effectiveness of Hadoopbased analysis and ease-of-use of the MapReduce technique in parallelization of the many data analysis algorithms. Another benchmark of SQL-like Big Data technologies is presented in (Grover et al., 2015), which uses queries that involve table scans, aggregations and joins. When comparing Hive, Presto, Drill, and Spark they conclude that Presto has outstanding runtime on performance over other big data solutions and that SparkSQL has an edge for analytics/machine learning.

In (Jethro, 2016) Hadoop Hive proprietary and Open Source tools alternatives are presented, analysing each one against the needs of Big Data processing, but providing few details about the tools. In (MapR, 2017) characteristics like latency and ANSI SQL completeness, are used to evaluate Drill, Hive, Impala, and SparkSQL. It is highlighted that Hive has low maintenance and is simple to learn, but not suitable for real-time queries; Impala has lower query latency, but memory errors are very frequent.

In (Santos et al., 2017), the authors use a denormalized TPC-H schema testing it in a low cost cluster with Hive, Spark, Presto, and Drill. The results show that it's possible to achieve adequate query execution times on modest hardware.

Although a comprehensive set of works in this field is already available, most of them are focus on extensive and sometimes confusing theoretical research, in some cases showing practical approaches and benchmarks in order to demonstrate tools performance. This work provides a clear and direct overview of a set of selected Big Data querying tools through the analysis of the state of the art, describing and comparing their characteristics and architectures in order to conclude in what scenario each tool should be used. We also perform an experimental evaluation using the TPC-H benchmark on Drill, Hawq, Hive, Presto, and Spark and also on Impala, which is very popular between users, but not compared in the previous works with all the tools referenced, maybe due to the fact that it requires the deployment of distinct Hadoop distribution.

BIG DATA PROCESSING TOOLS

The development and use of highly distributed and scalable systems to process Big Data is considered as one of the recent key technological developments. Some of these systems have proposed proprietary query languages or application program interfaces, while others have recognized the benefits of using SQL. Not all SQL analytical tools are equal, and that makes a challenge of picking the right tool. For comparison and due to their popularity, this work analyses and compares Drill, Hive, Impala, Presto, and Spark.

Drill

Drill is an open-source distributed system that supports data-intensive distributed applications for interactive analysis of large-scale datasets. It was developed with the goal of providing low latency and faster interactive queries. Drill supports several data storage NoSQL databases like

HDFS, Hive or HBase and SQL to query data, being capable of performing mathematical and statistical functions, string and dates manipulation. Like Hive, it allows creating custom functions in JAVA code. With the simple installation, it can scale-up to a very large cluster with thousands of nodes.

On top of all these characteristics, it offers connectivity and compatibility with Business Intelligence tools like Tableau, Microstrategy, Qlikview and Tibco. As core it has the 'Drillbit' service, which is responsible for accepting requests from the client, processing the queries, and returning results to the client. Drillbit can be installed and run on all of the required nodes in a Hadoop cluster, forming a distributed cluster environment, using ZooKeeper to maintain cluster membership, healthcheck information and identifying the appropriate nodes to execute query fragments.

The Drillbits are composed by components like: i) RPC end-point, allowing communication with the clients and receiving queries through the RPC protocol; ii) SQLParser, optimizing queries and generating a distributed query plan that is optimized for fast and efficient execution; and, iii)Optimizer, responsible for managing standard database optimizations, providing a distributed query plan for efficiently queries execution across different nodes.

In short, Drill provides interactive query capabilities enabling traditional business intelligence and analytics from different sources, suited for advanced analytic workflows, offering response times that vary between milliseconds to minutes depending on the query complexity (Gessler, 2014).

HAWQ

HAWQ, Hadoop With Query, recently brought into the Apache Foundation from the wellestablished Pivotal HAWQ, is a Hadoop native SQL query engine that combines the key technological advantages of Massive Parallel Processing (MPP) with the scalability and convenience of Hadoop, offering good performance. This parallel SQL query engine built on top of the HDFS, claims to be the world's fastest SQL engine on Hadoop (Lynn, 2016). It adopts a layered architecture and relies on HDFS for data replication and fault tolerance. HAWQ relies on both the PostgreSQL database and the HDFS storage as its backend storage mechanism, meaning that HAWQ can support the full ANSI SQL syntax. Another capability of HAWQ is its integration with MADlib, providing machine-learning capabilities directly in SQL.

Along with the Pivotal Extension Framework (PXF), HAWQ can work with data from several data sources like HBase, Hive, Text, Avro and Parquet, being able to run and be managed on Hortonworks HDP. The architecture of the HAWQ engine includes the following main components: I)HAWQ master: entry point responsible for accepting the connections from the clients and manages the system tables that contain metadata information about HAWQ itself, being also responsible for parsing and optimizing the queries and generating the query execution plan; II) HAWQ segments: represents the processing units, responsible for running the local database operations on their own data sets; iii) HAWQ storage nodes: used for storing all the user data (HAWQ relies on a proprietary file format for storing the HDFS data); iv)HAWQ interconnect: Responsible for managing the interprocess communication between segments during query execution.

Comparing with the other presented tools, it is easy to say that HAWQ has the most elaborate architecture. Queries are executed via PostgreSQL client and are sent to a HAWQ master, there they are parsed, optimized, fragmented and dispatched to HAWQ segments across the nodes for execution. During query execution YARN takes care of all resource management matters for overall performance and cluster stability. For the high availability of the master node, a standby master instance can be optionally deployed on a separate host (like HDFS Namenode and Secondary Namenode). The standby master host serves as backup when the primary master host becomes

unavailable. HAWQ Master also has a fault detector that checks the health of all segments periodically.

Hive

Hive is a system that supports the processing and analysis of data stored in Hadoop. It is considered to be the first to support SQL-on-Hadoop and has been used by many organizations such as Amazon. Built on the Hadoop platform, it supports familiar relational database concepts such as tables, columns, and partitions and includes some SQL support for unstructured data (Floratou et al., 2014). Hive gives structure onto data and perform ad-hoc queries and analysis using HiveQL, a SQLlike query language.

The query language of Apache Hive, HiveQL, has its own limitations, since some SQL features not yet available such as update and delete queries, and also, several restrictions on the use of subqueries. Being frequently considered as a high-latency system, oriented to batch workloads instead of interactive querying, Hive has been target of constant development to improve its performance.

Out of all these improvements we highlight the Stinger initiative that introduced ORC, a columnar format providing high compression and high performance and the execution engine Tez that optimizes Hive job execution which aims to Hive's latency caused by MapReduce.

Hive works by storing the metadata in Hive Metastore, used to provide a schema on read functionality and semantics check on queries (Owl, 2015a). Using the information available in the Metastore the Hive server transforms HiveQL queries into MapReduce jobs, enabling users to focus on specifying queries and how they will perform.

In terms of processing, SQL queries are submitted to Hive and executed as follows: i) Hive compiles the query; ii) an execution engine, like Tez or MapReduce, executes the compiled query; iii) the resource manager, YARN, allocates resources for applications across the Hadoop cluster; iv) the data manipulated in the query is maintained in HDFS; and, v) query results are made available over a JDBC/ODBC connection.

Summarizing, Hive has a good interface for anyone from the relational database world, demanding low maintenance and being simple to learn. We can think on Hive as a standard, it was the first to support SQL-on-Hadoop and it is included in all Hadoop distributions.

In the first distributions, Hive relied on Hadoop's MapReduce suffering from poor performance, being more appropriate for large scans, where query performance may not be so critical. It was not appropriate for On-Line Analytical Processing (OLTP) tasks. It is worth mentioning that using MapReduce is the main criticisms made to Hive, as the conversion to MapReduce jobs leads to higher query latency.

Impala

Impala is an open-source, state-of-the-art Massive Parallel Processing (MPP) SQL query engine designed for performance, real time, low latency and high concurrency processing. It was developed in C++ and Java, with the objective of combining the SQL support and multi-user performance of a traditional analytical database with the scalability and flexibility of Apache Hadoop. Impala implements a distributed architecture that can run on hundreds of machines in an existing Hadoop cluster. This tool can use two storage systems, HDFS or HBase, being possible to query data in both. It can also be integrated with Business Intelligence tools like Tableau, Pentaho, Micro Strategy and Zoom Data. Impala has been built to extend the key components of Hive, e.g., SQL syntax (HiveQL, meaning that like Hive it supports relatively little DML, there is no UPDATE or DELETE statements), metadata and schemas (Silva, Almeida, & Queiroz, 2016). Although it runs

natively on Hadoop, Impala doesn't use Hadoop to run the queries, instead it relies on a set of daemons installed on each DataNode tuned to optimize the local processing and avoid bottlenecks.

Impala relies on in-memory join implementations, meaning that queries can fail if the joined tables cannot fit into memory (Sakr, 2014). This is a major drawback, but usually real-life Big Data processing scenarios are built on top of Hadoop clusters, containing high amount of RAM memory. If the amount of memory available ensures that data that is being processed by queries fits, inmemory processing can provide outstanding performances.

In summary, Impala has a large advantage since processing is done in memory, reducing latency and Disk IO, especially in real-time and ad-hoc queries, as long as the runtime is short enough that node failures during the query execution are unlikely and the data involved in the query fits in memory.

Presto

Presto is an open source distributed SQL query engine for running interactive analytical queries against data sources of all sizes ranging from gigabytes to petabytes, targeted at analysts who expect response times ranging from seconds to minutes. It was developed by Facebook, making available to give Hadoop some SQL-like capabilities, being optimized for low latency and interactive query analysis.

Facebook uses Presto for interactive queries integrating several internal data stores, including its 300PB data warehouse. Over 1,000 Facebook employees use Presto daily to run more than 30,000 queries, each scanning over a petabyte per day (Facebook, 2016). Using in-memory processing, instead of MapReduce, it avoids side steps, unnecessary I/O and latency, allowing faster response times. As a result, Facebook claims that Presto runs 10 times faster than Hive (Grover et al., 2015).

Presto can perform simple queries in few hundred milliseconds and more complex in few minutes. Running on a cluster of machines, it operates on an architecture that includes a coordinator, multiple workers and a client. The client Presto CLI, is a terminal-based interactive shell that submits SQL statements to a coordinator to get the result. Presto Coordinator, Parses the SQL queries for defining the query execution plan. Finally, Presto Workers, receive assignments from the coordinator and delivering results.

Spark

Spark is a highly distributed processing framework that provides an ease of use tool for efficient analytics on heterogeneous data. It was originally developed in 2009 in UC Berkeley's AMPLab, and open sourced in 2010 as an Apache project, running on top of Hadoop Distributed File System (HDFS). Its main characteristics are the way it works, loading the data into a cluster's memory and performing the necessary queries.

Spark provides full access and compatibility with Hive existing data and uses the concept of Resilient Distributed Datasets (RDD), which describes an immutable collection of objects that are partitioned and distributed across multiple nodes of a cluster, allowing parallel processing. One of the key modules of Spark is Spark SQL, used for processing structured data with DataFrames (equivalent to relational database tables) that organize data in named columns. To extend the vocabulary of SparkSQL, it is possible to plug-in custom user defined functions (UDFs). Spark uses a master/worker architecture, where a single coordinator, the master, manages all workers, executing tasks, as depicted.

COMPARING BIG DATA PROCESSING TOOLS

To determine the best querying tools, in this section we present a set of characteristics and requirements that every structure should fulfill to successfully store and process Big Data. Based on

the works of [9], [10] and [13] the following characteristics can be defined as the main requirements for an efficient Big Data processing architecture:

- *Scalability*: Scaling is the ability of the system to adapt to increased demands in terms of data processing; linear scalability is necessary for the explosive growth of data size;

- *High throughput* (processing speed): Big Data's velocity demands that data be ingested and processed at high speeds, this requiring an infrastructure that is extremely fast across input/output (I/O), processing, and storage;

- *High degree of parallelism*: By processing data in parallel, we can distribute the load across multiple machines, each one having its own copy of the data, but processing a different part;

- *Programming language support*: The support of programming languages can be useful for integrating different BI tools or for supporting other developments;

- *SQL Support*: Since SQL is the base for queries processing;

- *Distributed Architecture*: Distributed processing across several servers (nodes), providing parallel computing;

- *Fault Tolerance*: Mechanisms for detecting failures and for recovering from them;

- *Single Point of Failure*: Identifying when the whole query is aborted if one of the processing nodes fails;

- *Machine Learning Algorithms/Tools*: For advanced data processing, allowing the identification of hidden patterns and trends in Big Data.

Considering the querying engines and their core features presented in the previous section, Table 1 shows the key evaluation characteristics for comparing these tools.

Table 1 Big Data Analytical Tools Comparison

Feature	**Drill**	**Hawq**	**Hive**	**Impala**	**Presto**	**Spark**
Owner	Community	Greenplum	Community	Cloudera	Facebook	Community
Cluster Size Limit(node)	Thousands	Thousands	Hundreds	Thousands	Thousands	Thousands
SQL Support	ANSI SQL	ANSI SQL	HiveQL	HiveQL	ANSI SQL	ANSI SQL (limited) & HiveQL
Latency	Low	Low	Medium	Low	Low	Low
Data Size Limit	Gigabytes to Petabytes		Terabytes	Gigabytes to Petabytes	Gigabytes to Petabytes	Terabytes
Scalability	High	High	High	Very High	Very High	High
Machine Learning	No	Yes	No	Yes	No	Yes

Data Visualization	No Native support, compatibility with BI tools	No Native support, compatibility with BI tools	No Native support, compatibility with BI tools	No Native support, compatibility with BI tools	No Native support, compatibility with BI tools	No Native support, compatibility with BI tools
Processing Speed	Fast	Very Fast	Slow	Very Fast	Very Fast	Fast
Language Support	C++, Java and other languages JDBC/ ODBC	Python, Perl, Java, C/C++, R	Java	All languages supporting JDBC/ ODBC	C, Java, Node.js, PHP, Python, R, Ruby	Java, Python, R, Scala, JDBC/ ODBC
Use Cases	Real-time Interactive queries and analysis/BI	Batch processing and interactive queries	Batch processing	Real-time Interactive queries and analysis/BI	Real-time Interactive queries	Batch and stream processing, interactive queries and ML
Distributed Architecture	Yes	Yes	Yes	Yes	Yes	Yes
Single Point of Failure in Query Execution	No	Yes	Yes, if failure at the master node.	Yes, if any host quits query execution	Yes, if any host quits query execution	No
Fault Tolerance	Yes	Yes	Yes	No	No	Yes
File/Storage Formats	CSV, TSV, PSV, Parquet, Hadoop Sequence Files (Key-value pairs)	Text, Parquet, Avro, CSV	ORC, AVRO, Parquet, and Text	Parquet, Text, Avro, RCFile or Sequence files	Text, Sequence Files, RCFile, ORC and Parquet	JSON, Parquet, CSV, ORC, RDDs, Hive Tables and External Databases

Comparing Impala and Hive, both support SQL but Hive is more appropriate for long-running batch processing, offering robustness and low maintenance. Impala provides a better support for real-time analytic queries having less latency than Hive since it bypasses MapReduce. Using Hive with Tez instead of MapReduce makes Hive a possible solution for real-time analytic/ad hoc queries and OLTP environments, but based on some benchmarks performed previously (e.g (Devadutta Ghat, David Rorke, 2016), (Kornacker et al., 2015)), Impala can provide better response times.

In terms of fault tolerance, Hive easily retries queries in case of failure, which does not happen in Impala. Spark has a machine-learning library (MLib), which makes it suitable for analytics, being an adequate choice when the objective is not just querying data but working with it in an exploratory manner, even for real-time data streaming. Spark has short query response time, being faster than Hive, even so, Impala seems to be faster (Devadutta Ghat, David Rorke, 2016),(Kornacker et al., 2015).

Drill suits best when the main objective is only queried data from several data sources. It has better response times than Spark, which does not happen in the case of a single source. It is also fully ANSI SQL compliant. Another suited solution for combining data from multiple sources is Facebook Presto, performing queries with concerns with CPU efficiency and latency, having faster response times than Drill. In some works, Cloudera claims that Impala is 5.3 to 7.5 times faster than Presto (Owl, 2015b). Of all the presented tools, after the analysis of the state of the art, HAWQ is the less known option. However, it can provide strong processing with full ANSI SQL support and provides machine learning and data mining algorithms. On several benchmarks, HAWQ managed to achieve performance improvements of 10x to 600x (Morgan, 2013), surpassing Impala, improvements that could turn batch systems into interactive ones.

EXPERIMENTAL EVALUATION

The first contact with these kinds of tools was made using only a personal laptop, which was unfeasible, since Big Data applications stress all system components, such as CPU cores, memory, storage and network I/O, too heavy for a single machine. To support distributed parallel processing and boost the speed of data analysis applications we use a cluster with 4 nodes. After the cluster configuration we installed Cloudera and Hortonworks Hadoop distributions. We need these two distinct distribution since Impala only runs on Cloudera distribution, the remaining tools are compatible with Hortonworks, this need to install a different distribution. In this section we describe how we performed the experimental evaluation in the previously analyzed tools, using the TPC-H benchmark, detailing some aspects of the experimental setup.

An overview of the TPC-H Benchmark

The TPC Benchmark™H (TPC-H) is a decision support benchmark that provides a suite of business oriented queries, illustrating decision support systems that examine large volumes of data through the execution of queries with a high degree of complexity, using a variety of SQL operators, giving answers to common analytics scenarios and critical business questions (Council, 2017). Using the TPC-H benchmark requires the generation of the data schema and the population of the corresponding data, being afterwards possible to run a set of 22 business oriented ad-hoc queries over a database schema, which has eight tables. More details regarding the TPC-H benchmark can be found at the official website (http://www.tpc.org/tpch) and on (Council, 2017), where it is possible to find the DBGen tool, used for generating the several needed datasets for the benchmarks.

Experimental setup

As previously mentioned, the Cloudera and Hortonworks Hadoop distributions were installed in the cluster, which simplifies the configuration of all tools. The cluster integrates 4 nodes, 1 acting as Master and the remaining as Workers, that in the Hadoop distribution, HDFS will be installed with one Namenode (master node) and three Datanodes (worker/slave nodes), all accessible through SSH (see Figure 1). In short, the hardware configuration of each node includes one octa-core CPU 1.80GHz, 16 GB of RAM and one SATA disk with sizes ranging between 120 and 240 gigabytes, connected through gigabit Ethernet (1000 megabits per second) to achieve the gigabit data rate, with 64-bit CentOS Linux 7 as the operating system. As seen frequently in the literature, SQL-onHadoop systems need significant amounts of RAM to process data, benefiting from the availability of a higher number of nodes available (e.g. cluster used in (Floratou et al., 2014) with 21 nodes and 96 GB of RAM. Even though, as we see in (Santos et al., 2017), it's possible to perform this kind of experiences on low cost cluster.

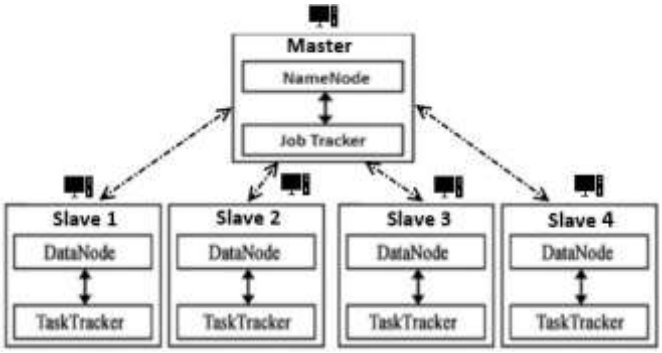


Figure 1 Hadoop Master/Worker Architecture

Performance Evaluation

To measure the query performance of big data analytical tools, we run all 22 TPC-H queries, 5 times each, and attempt to eliminate time variations by cleaning file system cache before running each query. The final query execution time is the average of the 5 runs. We also the total time execution for all queries and also the speedup as lowest_tool_time/current_tool_time.

We perform our performance evaluation using 10, 30 and 100 GB dataset to observe scalability effects. For this benchmark, the recommendations of the Stinger initiative were followed, and the data was stored in ORC format, except for Impala that does not support ORC, requiring the use of the parquet format. We chose these file formats high focus on efficiency leads to some impressive compression ratios. The work (Shanklyn, 2017) shows the sizes of the TPC-DS dataset at Scale 500 in various file formats, including ORC and Parquet, where data in ORC gets 78% smaller than the original size. With Impala, 62% smaller than the original size. We confirmed the high compression rates when generating out benchmark data. Since our cluster's capacity is limited, it is relevant to choose a file format that benefits higher compression rates.

Also, the Tez execution engine when using Hive. The number of rows of TPC-H schema tables according to the Scale Factor (SF) is shown in Table 2, where we can see largest tables are Lineitem, Orders and Customers.

Table 2 TPC-H Number of rows with different SF

SF of TPC-H	**Customers**	**Lineitem**	**Nation**	**Region**	**Orders**	**Supplier**	**Part**	**Partsupp**
10GB	15 000 000	60 000 000	25	5	15 000 000	100 000	2 000 000	8 000 000
30GB	45 000 000	180 000 000	25	5	45 000 000	300 000	6 000 000	24 000 000
100GB	150 000 000	600 000 000	25	5	150 000 000	1 000 000	20 000 000	80 000 000

The results for TPC-H with scale factors of 10, 30 and 100 GB are represented Table 3 Table 4 Table 5 In overall, we can observe that queries with complex joins and subqueries across several tables are the operations that are more demanding in terms of resources, requiring more processing time (e.g Q2, Q5, Q7, Q8, Q9, Q10, Q18 and Q21). Other costly operations include aggregations operations and arithmetic calculations present in several queries (e.g Q1, Q10, Q13, Q18, Q20, Q21) and large IN clauses against a series of at most eight constant values (e.g Q12, Q16, Q19 and Q22).

As the scale factor (SF) grows, the used system starts to reach its limit, taking too long to process some queries or being, in some cases, unable to finish them, mainly for Presto as can be seen in Table 5.

Regarding the results for the 10GB SF, we had no problem running all the queries, with Hawq and Presto as the fastest tools with a speedup of 5.79 and 5.18, which means these tools are 5 times faster than Spark. Impala performed queries ranging 1.2s to 36s, and Hawq ranging from 1.15s and 19s. After Impala, Drill performed relatively well in the overall, except for Q19 and Q21 where the times were similar to Spark.

Hive performed better than Spark for most of the queries, except for queries Q7 and Q21. In particular, Q21 is the one that requires more processing time, for all the workloads and all tools, since it involves a join across four tables (Supplier, Lineitem, Orders and Nation), being the most complex query of the benchmark (the joins also include the two largest tables (Lineitem and Orders).

Table 3 Query Execution Time for 10 GB (in seconds)

	Drill	**Hawq**	**Hive**	**Impala**	**Presto**	**Spark**
Q1	8.83	5.88	17.80	2.87	4.49	26.73
Q2	12.69	6.13	23.31	2.66	10.98	36.21
Q3	8.18	6.50	20.49	11.83	7.61	32.72

Q4	6.77	5.34	34.39	10.62	6.05	60.18
Q5	13.21	10.08	23.33	19.50	*10.6*	34.54
Q6	4.61	3.49	10.84	1.23	2.66	18.82
Q7	20.36	5.94	51.26	22.95	12.59	46.51
Q8	13.10	4.60	23.32	31.62	10.23	31.84
Q9	21.36	11.06	36.77	35.77	16.16	52.97
Q10	11.12	7.48	32.30	6.28	6.78	44.02
Q11	3.77	3.92	16.00	1.15	4.17	23.42
Q 12	8.53	1.15	14.82	2.21	4.03	25.14
Q13	6.42	4.52	26.39	10.23	5.97	68.68
Q14	6.49	3.92	13.86	1.78	2.87	23.39
Q15	12.97	5.52	19.47	4.25	3.16	30.51
Q16	12.70	6.46	17.78	1.51	5.88	30.28
Q 17	9.20	15.62	16.22	1.68	8.41	23.85
Q18	32.77	14.83	42.21	12.18	14.40	86.33
Q 19	31.01	2.21	20.51	3.16	4.41	33.85
Q20	11.52	7.93	19.89	8.98	3.87	26.79
Q21	96.39	18.99	128.08	32.24	25.18	106.92
Q22	7.18	4.29	25.21	7.95	3.66	38.94
Total	359.18	155.86	634.25	232.65	174.16	902.64
Speedup	*2.51*	*5.79*	*1.42*	*3.88*	*5.18*	*1*

For some of the queries, requiring more processing time or presenting more differences between the fastest tools in terms of the time required to process the data, Figure 2 shows the three fastest tools (Hawq, Presto and Impala) with a represented query set (Q5, Q7, Q9 , Q17, Q18, Q21 and Q22). Presto and Hawq run with similar performance, surpassing Impala, except on Q17, a relatively simple query where Impala was the fastest.

In Q17 we also see that Presto performs the query about 2x faster than Hawq, this query has some aggregation operations, but the heaviest workload resides on the correlated subqueries and for this query Impala process it more quickly. We see that Impala was also faster than the other two tools in Q18, with a 2s seconds difference, if we look at this query we see that the heavier workload resides on the aggregation calculations and joins, but since data fits in the available memory it could keep up with Presto and Impala.

For the remaining queries, we see that Impala needs much more time to execute the queries, the execution of Q8 and Q9 registers the more significant differences due to a subquery that includes JOIN operations with 6 of the 8 tables several tables (including Lineitem and Orders).

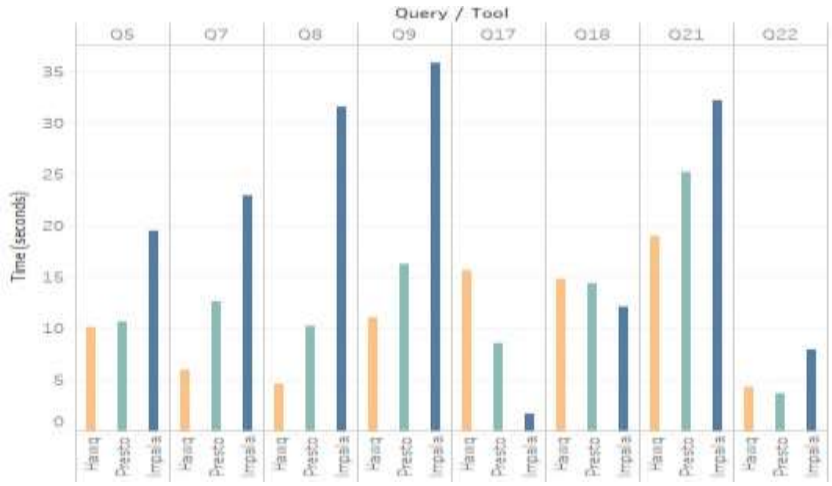


Figure 2 Sample Query Set for 10 GB between fastest tools (Hawq, Presto, and Impala).

Regarding the 30 GB SF results presented on Table 4, the execution times took longer, but not linearly, meaning for 30GB, the execution time wasn´t 3x the execution times of 10GB. For Hawq, Impala and Presto, considering the total time required to run the 22 queries we see an decrease in the speedup of about 2.42, 2.66 and 1.97 comparing it with the 10 GB SF, meaning the overall performance of these tools improved.

For Q1, which computes eight aggregates: a count, four sums and three averages Drill was the slowest tool due to its performance taking 2s more than Spark.

Also on Q2, besides performing multiple joins, it includes a LIKE string manipulation and a subquery that calculates minimum cost part supplier for a certain part, Drill was once again the slowest. On other queries, Drill could keep up with fastest tools performing similarly in some queries like Q3, Q4, Q5, Q11, Q13.

We also notice that Presto start to be the quickest tool in more queries, with Impala and Hawq maintaining good performance in most queries, except for Q21 that we were unable to run on Presto, and for the remaining with times between 72.63 for Impala and 213s.

We also notice that hive manages to be faster than Impala in some queries like Q3, Q5 and Q7.

Table 4 Query Execution Time for 30 GB (in seconds)

	Drill	**Hawq**	**Hive**	**Impala**	**Presto**	**Spark**
Q1	30.93	15.2	25.78	13.21	13.10	28.82
Q2	58.12	13.32	36.86	11.36	13.43	38.47
Q3	20.97	24.47	29.72	52.50	16.72	36.75
Q4	18.96	21.01	85.66	51.49	12.31	125.66
Q5	28.79	26.67	45.00	68.46	27.59	49.29
Q6	7.70	8.83	10.24	5.57	3.47	22.61
Q7	42.67	13.66	76.40	78.68	34.18	82.52
Q8	17.19	12.21	32.56	12.11	26.68	41.86
Q9	27.68	25.22	182.58	164.2	37.75	126.68
Q10	27.72	23.26	62.17	17.72	14.36	63.17
Q11	4.22	8.88	23.68	6.88	9.14	52.15
Q 12	14.86	20.11	29.65	5.55	11.28	27.59
Q13	12.97	7.92	54.60	30.84	9.83	62.82
Q14	12.95	19.14	21.02	5.85	6.51	33.83

Q15	25.8	13.46	24.18	3.93	6.68	38.66
Q16	14.84	10.05	25.45	2.77	7.04	30.70
Q 17	11.21	44.22	22.92	3.88	21.12	30.50
Q18	52.38	38.73	130.77	41.24	44.55	97.68
Q19	52.21	4.67	122.36	7.06	12.24	133.51
Q20	21.28	16.75	23.12	13.81	8.47	36.77
Q21	129.71	72.63	145.69	213.38	-	176.85
Q22	12.73	8.00	34.81	22.59	6.96	34.81
Total (w/Q21)	516.18	375.78	1099.53	619.7	343.41	1194.85
Speedup	*2.31*	*3.18*	*1.01*	*1.93*	*3.48*	*1*

Taking now a different set of queries, for the three tools that present an overall better performance for 30 GB (Presto, Hawk, Drill), Figure 3 shows that Impala keeps performing times with a significant longer duration for this set of queries, except on Q19, Q17, Q14 (comparing with other queries, one of the differences is the absence GROUP BY and ORDER BY clauses and have less aggregation operations workload). Regarding presto we can see that it starts to surpass Hawq more often, still Hawq performs better in queries that have more complex JOIN operations and subqueries like Q7, Q8, Q9, Q13 and Q19 for this SF, also it was able to run the most complex query 21, being about 3x times faster than Impala. We can see that in Q3 Impala was surpassed by Hive mostly due to the heavy performance of aggregation operations ORDER BY and GROUP BY.

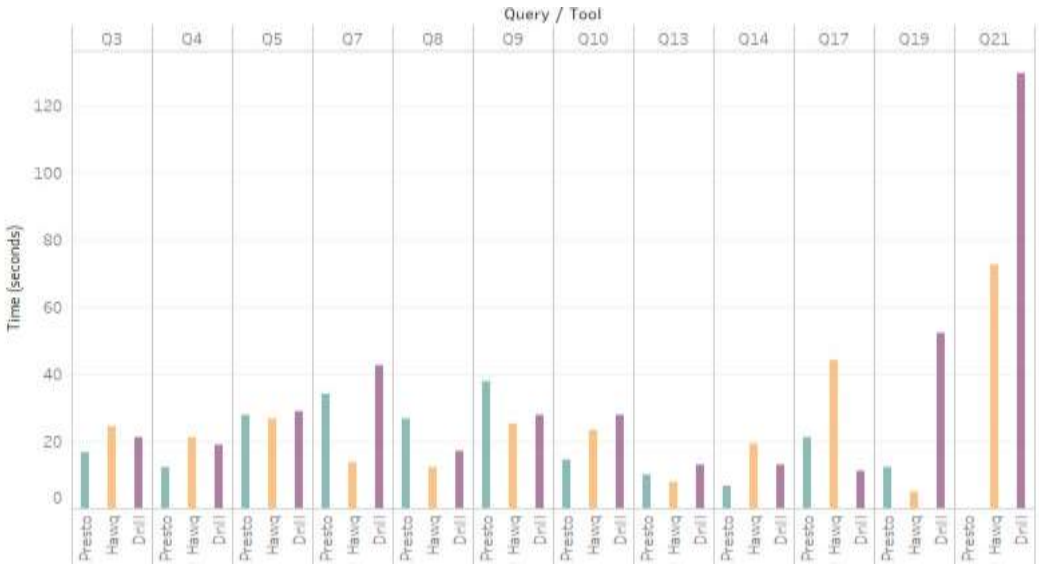


Figure 3 Sample Query Set for 30 GB between fastest tools (Hawq, Presto, and Impala)

In the experiments when we reach 100GB the limit in terms of resources for our cluster is achieved. In this case as we can see in Table 5, Drill was unable to process Q18 and Q21, while Presto was unable to run 7 queries (Q2, Q7, Q8, Q9, Q18 and 21) due to JOIN due to the voluminous

JOIN operations. As Presto uses main memory to process the data (in-memory processing), and since JOINS present in the queries process a considerable volume of data, as soon as the tool gets out of memory, the processing stops.

Although Presto failed in many queries, it is worth mentioning that it was the fastest tool in almost all remaining queries (Q1, Q3, Q4, Q6, Q10, Q12, Q13 and Q14), leading us to believe that it

in a cluster with more resources (RAM memory), it would be a suitable tool for interactive querying. Regarding Drill, like in the previous kept perform similarly to the fastest tools in some queries (e.g Q4, Q5, Q16, Q18, Q19), meaning it's not a tool we should completely ignore although its surpassed largely by other tools in some queries. At this SF, we also notice that Spark does not present the slowest processing times (slowest only in Q11 and Q16), as in the 30GB SF.

For HAWQ, increasing the workload had a huge negative impact in its performance, having the longest times in several queries (Q1, Q6, Q12, Q14 and Q20), on the other hand, it was the fastest one in the most complex query (Q21).

We notice that although Spark and Hive have not achieved the best results, they still managed to run all queries, leading us to acknowledge their robustness. In some queries, these tools present performances that are similar to the ones presented by Hawq and Impala, but never with the times achieved by Presto.

As for Impala, similar to presto, is very efficient if the processed data fits in the available memory. Otherwise, it applies a feature called "spill to disk", preventing out-of-memory errors. In those cases, data is written in disk and the query does not crash, although it greatly impairs the performance, which the huge time differences that we see along the benchmark for Impala.

Table 5 Query Execution Time for 100 GB (in seconds)

	Drill	Hawq	Hive	Impala	Presto	Spark
Q1	115.32	157.61	40.82	38.42	34.41	52.24
Q2	154.18	38.87	155.31	36.17	-	116.4
Q3	102.79	185.82	232.34	289.18	55.3	150.08
Q4	97.06	201.68	123.18	327.33	24.75	105.16
Q5	113.03	171.99	201.86	315.96	-	178.07
Q6	34.7	135.87	38.8	23.22	12.89	40.01
Q7	165.29	162.34	229.77	350.46	-	191.04
Q8	79.02	140.96	129.23	566.02	-	123.51
Q9	86.47	217.33	442.21	1070.66	-	420.5
Q10	98.55	150.81	259,13	60.52	46.04	143.69
Q11	16.35	28.65	31.63	10.64	29.78	44.09
Q 12	50.62	157.27	46.11	22.69	21.38	52.45
Q13	56.63	44.32	134.84	142.96	26.66	121.48
Q14	46.04	139.81	96.95	33.03	15.17	69.56
Q15	67.56	131.61	131.96	18.26	23.01	74.98
Q16	19.99	42.24	49.15	8.19	11.86	54.38
Q 17	46.4	396.57	69.91	8.82	86.06	61.67
Q18	-	280.87	373.75	226.01	-	262.55
Q 19	254.98	126.38	449.44	32.79	19.76	409.31
Q20	49.78	149.11	59.37	21.73	22.77	59.51
Q21	-	453.56	2870.82	1503.29	-	2869.76
Q22	17.2	41.84	96.19	62.82	14.37	76.44
Total	-	3555.51	6262.77	5169.17	-	5676.88

Speedup	*-*	*1.76*	*1*	*1.21*	*-*	*1.10*

Taking now some of the queries that all tools could process, Figure 4 shows how the query processing time is influenced by the type of query, as some tools are the fastest ones in some queries and can be the slowest ones in others. For this query set, we observe that for simple selection queries like Q1 or Q6, HAWQ seems to lose its edge being about 3x times slow by Hive and Spark, with Presto and Impala performing similarly to Hive.

When heavy aggregation operations are involved, the execution time is affected, but it Is when we perform JOIN operations on huge volumes and subqueries that the tools really grasp to perform. In these cases, in-memory performance engines like Presto and Impala seems to lose its advantage if the data processed by the query doesn't fit in memory.

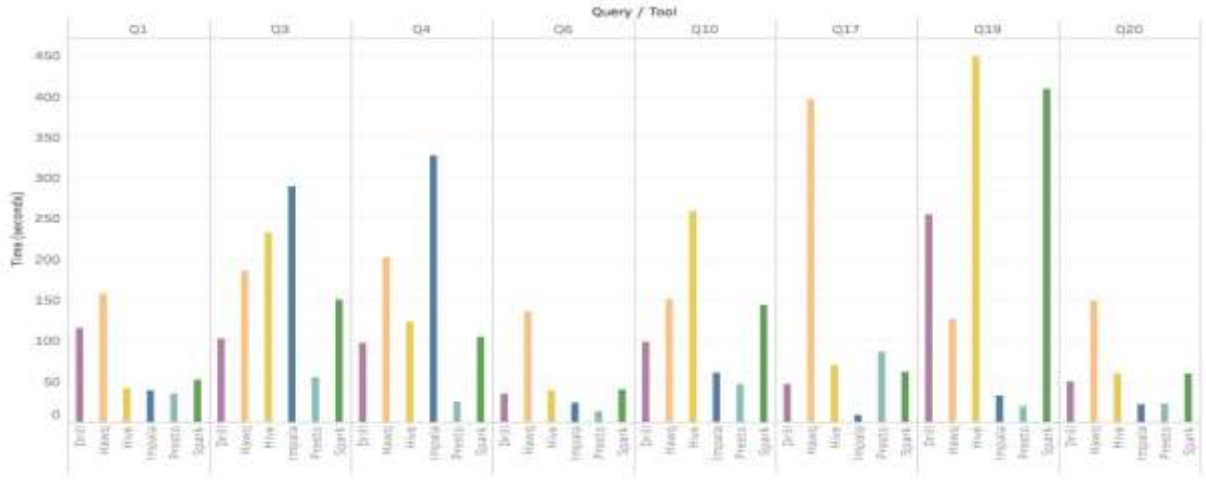


Figure 4 Sample query set for 100GB

After comparing the results obtained we find that Hive and Spark are less suitable when querying in low data volume, due to the long coordination time of jobs, but if we look at the different SF, as the data volume grows, they seem to catch up (less time difference between tools for the same queries) and even surpass the fastest tools on some queries, as we can see in Table 5 (where Hive and Spark surpassed mostly Impala and in some cases, Hawq in several Queries). Another factor that worth to point out, is the SQL compatibility, in Table 1 we referenced the level of SQL compatibility of the several tools, where we can see that Drill and Presto have more extensive SQL compatibility, for the remaining tools that use HiveQL.

This problem emerged when we were running the benchmark queries, mostly in cases that involved subqueries. To surpass this, we had to rewrite the queries with subqueries in the WHERE HiveQL supports clause, since not all types on subqueries on FROM, WHERE and HAVING clauses. In these cases, we replace the subquery by SQL Views to store the data and then use that view in the original query instead of the subquery (this change was made for Q2, Q11, Q15 and Q22), an additional work that add to be do not to run the queries.

CONCLUSIONS AND FUTURE WORK

Big Data processing tools can deal with the massive quantity of data that exists nowadays, allowing users to perform ad-hoc querying.

In this work, the characteristics and architecture of big data processing tools were described and we can consider that there is no one-size fits all SQL-on-Hadoop tool. In our opinion supported by

the theoretical investigation and the performance evaluation using TPC-H benchmark, in terms of query execution time, the best tools are Presto, Impala, and HAWQ (in this order), followed by Apache Drill. These are the tools that show the best performance in the execution of TPC-H queries and consequently best speedup. Presto runs out of memory for some queries of 100 GB dataset, although in terms of performance it is the most suitable tool for interactive querying due to inmemory processing, reducing query execution time significantly with the advantage of being fully ANSI SQL compliant.

Comparing Presto, Impala, and Hawq, we consider that all of them have good performance, depending on query complexity. According to the experimental evaluation and the results of query execution time for the different SF, we presume that with more RAM memory Presto would outperform all the others. With the data volume increasing to 100GB, Hive and Spark start to give better results comparing with the tools that were faster with lower SF of 10GB and 30GB. Overall, we believe that Spark is more suitable when the objective is not just to query data, but to perform algorithmic analysis, statistics and Machine Learning.

As future work, because Presto is the tool that achieves better performance, we intend to add more memory or scale the cluster (adding more nodes) and confirm the results to the 100 GB SF or even scale the SF more. We also intend to use TPC-DS benchmark, where queries are more complex and use a different database schema model and observe if the analyzed tools maintain the same behaviour.

References

Bobade, V. B. (2016). Survey Paper on Big Data and Hadoop. International Research Journal of Engineering and Technology, 3(1), 861–863.

Council, T. P. P. (2017). TPC BenchmarkTM H Standard Specification Revision 2.17.2. Retrieved from http://www.tpc.org/tpc_documents_current_versions/pdf/tpc-h_v2.17.2.pdf

Devadutta Ghat, David Rorke, D. K. (2016). New SQL Benchmarks: Apache Impala (incubating) Uniquely Delivers Analytic Database Performance. Retrieved from https://blog.cloudera.com/blog/2016/02/new-sql-benchmarks-apache-impala-incubating-2-3uniquely-delivers-analytic-database-performance/

Facebook. (2016). Presto Overview. Retrieved from https://prestodb.io/overview.html

Floratou, A., Minhas, U. F., & Ozcan, F. (2014). Sql-on-hadoop: Full circle back to shared-nothing database architectures. Proceedings of the VLDB Endowment, 7(12), 1295–1306. https://doi.org/10.14778/2732977.2733002

Gessler, A. C. (2014). MapReduce to Couple a Bio-mechanical and a Systems-biological Simulation, (156).

Grover, A., Gholap, J., Janeja, V. P., Yesha, Y., Chintalapati, R., Marwaha, H., & Modi, K. (2015). SQLlike big data environments: Case study in clinical trial analytics. Big Data (Big Data), 2015 IEEE International Conference on, 2680–2689. https://doi.org/10.1109/BigData.2015.7364068

Jethro. (2016). Hadoop Hive and 11 SQL-on-Hadoop Alternatives. Retrieved from https://jethro.io/hadoop-hive

Kornacker, M., Behm, A., Bittorf, V., Bobrovytsky, T., Ching, C., Choi, A., Yoder, M. (2015). Impala: A Modern, Open-Source SQL Engine for Hadoop. CIDR(Conference on Innovative Data Systems Research). Retrieved from http://impala.io/

Landset, S., Khoshgoftaar, T. M., Richter, A. N., & Hasanin, T. (2015). A survey of open source tools for machine learning with big data in the Hadoop ecosystem. Journal of Big Data, 2(1), 24. https://doi.org/10.1186/s40537-015-0032-1

Lynn, D. (2016). SQL ON HADOOP THE DIFFERENCES AND MAKING THE RIGHT CHOICE. Retrieved from http://www.agildata.com/sql-on-hadoop-the-differences-and-making-the-right-choice/

MapR. (2016). SQL on Hadoop Details. Retrieved from https://mapr.com/why-hadoop/sqlhadoop/sql-hadoop-details/

MapR. (2017). SQL on Hadoop Details. Retrieved from https://mapr.com/why-hadoop/sqlhadoop/sql-hadoop-details/

Morgan, T. P. (2013). EMC morphs Hadoop elephant into SQL database Hawq. Retrieved from http://www.theregister.co.uk/2013/02/25/emc_pivotal_hd_hadoop_hawq_database/

Owl, C. (2015a). The SQL on Hadoop landscape: An overview (Part I). Retrieved from http://cleverowl.uk/2015/11/19/the-sql-on-hadoop-landscape-an-overview-part-i/

Owl, C. (2015b). The SQL on Hadoop landscape: An overview (Part II). Retrieved from http://cleverowl.uk/2015/12/25/the-sql-on-hadoop-landscape-an-overview-part-ii/

Prasad, B. R., & Agarwal, S. (2016). Comparative Study of Big Data Computing and Storage Tools : A Review. International Journal of Database Theory and Application , 9(1), 45–66. Retrieved from http://www.riss.kr/link?id=A101827491

Rodrigues, M., Santos, M., & Bernardino, J. (2017). Describing and Comparing Big Data Querying Tools. Recent Advances in Information Systems and Technologies. Advances in Intelligent Systems and Computing, 569, 115–124.

Sakr, S. (2014). A brief comparative perspective on SQL access for Hadoop. In Recent Advances in Information Systems and Technologies, Volume 1 (pp. 1–9).

Santos, M. Y., Costa, C., Galvão, J., Andrade, C., Martinho, B. A., Lima, F. V., & Costa, E. (2017). Evaluating SQL-on-Hadoop for Big Data Warehousing on Not-So-Good Hardware. Proceedings of the 21st International Database Engineering & Applications Symposium on - IDEAS 2017, 242–252. https://doi.org/10.1145/3105831.3105842

Shanklyn, C. (2017). ORCFILE IN HDP 2: BETTER COMPRESSION, BETTER PERFORMANCE. Retrieved from https://hortonworks.com/blog/orcfile-in-hdp-2-better-compression-better-performance/

Silva, Y. N., Almeida, I., & Queiroz, M. (2016). SQL: From Traditional Databases to Big Data. Proc. of the SIGCSE - ACM Technical Symp. on Computer Science Education, 6. https://doi.org/10.1145/2839509.2844560

Appendix C – CAISE'18 Submited Paper

Experimental evaluation of Big Data Analytical Tools

Mário Rodrigues[1[0000-0003-1065-414X]], Jorge Bernardino[2[1111-2222-3333-4444]] ,

Maribel Santos[3[1111-2222-3333-4444]]

[1] Polytechnic of Coimbra, Institute of Engineering of Coimbra (ISEC), Coimbra, Portugal
a21190357@alunos.isec.pt
[2] Polytechnic of Coimbra, Institute of Engineering of Coimbra (ISEC), Coimbra, Portugal
CISUC – Centre of Informatics of University of Coimbra, Coimbra, Portugal
jorge@isec.pt
[3] ALGORITMI Research Centre, University of Minho, Guimarães, Portugal
maribel@dsi.uminho.pt

Abstract In the past years, Big Data has become a hot topic across several business areas. Because of the tremendous increase of the Internet usage, like social media, daily online operations and transactions from multiple sources, the amount of generated data has grown exponentially. Despite the complexity and the many challenges associated to the use of the Big Data concept, we cannot ignore the potential lying in it, and so, efficient querying analysis mechanisms are mandatory for data analysis purposes. Due to the extensive use of SQL, the number of SQL-on-Hadoop systems has increased significantly, transforming Big Data Analysis in a more accessible practice, allowing users to perform ad-hoc querying and analysis. Therefore, there is much discussion, about querying tools and, specifically, those more appropriated for specific analytical needs. This work performs a performance evaluation of popular Big Data analytical tools Drill, Hive, HAWQ, Impala, Presto, and Spark using TPC-H benchmark and TPC-DS benchmark.

Keywords: Big Data, SQL-on-Hadoop, Query Processing, Big Data Analytics.

1. Introduction

Big Data as a research topic is still in its infancy. Although, in the past years, Big Data has become a hot topic across several business areas. One of the main challenges regarding this concept is how to handle the massive volume and variety of data efficiently.

In this context, organizations are starting to realize the importance of taking advantage of data to support the decision-making process. Despite the complexity and the many challenges associated to the use of the Big Data concept, we cannot ignore the potential lying in it. It can support for analytics and for the identification of hidden patterns, which are very effective in defining business strategies.

In the Big Data scope, Structured Query Language (SQL) processing has gained significant attention, as many enterprise data management tools rely on SQL, and, also, many users are familiar and comfortable with it. For an organization, this opens up Big Data to a much larger audience, increasing the return of investment in this field [12]. This fact led significantly increase of the number of SQL-on-Hadoop systems. However, not all SQL querying tools are suited for the same scenarios, being a challenge the selection of the most appropriate one. Since there is many descriptive information about this kind of tools and their architectures, we select the ones that in our opinion represent the state of the art, without entering in detail. We evaluate the following popular big data analytical tools: Drill, HAWQ, Hive, Impala, Presto, and Spark. An experimental evaluation of Big Data analytical tools was performed using the TPC-H and TPC-DS benchmarks.

The remainder of this article is organized as follows. First, in section 2 we present the related work. In section 3 we describe the compared tools in a very summarized way. In sections 4 and we present a comparative table for the selected tools and perform a performance evaluation of analyzed tools. Finally, section 6 concludes the paper and present some future work.

2. Related Work

BigData is identified as one of the biggest IT trends of the last few years, which includes a large amount of work regarding querying and processing tools. With this growth, SQL processing, namely SQL-on-Hadoop, has been widely studied, analyzing and evaluating the performance and of several processing tools. In this context, the work performed on [1] provides a performance comparison of Hive and Impala using the TPC-H benchmark and a TPC-DS inspired workload, analyzing the I/O efficiency of their columnar formats. The results show that Impala is faster than Hive, on either MapReduce and on Tez for the overall TPC-H and TPC-DS benchmarks.

In [28], [29] the motivation for using Hadoop is presented, along with the strengths and limitations of tools like Impala Hive, BigSQL, HAWQ, and Presto. The work of [30] presents the results using TPC-DS queries, comparing response time for a single user and for 10 concurrent users, using Impala, Hive, and Spark, showing that Impala was the only engine that provided interactive query response on both user scenarios.

The work of [31]presents a good overview of the reasons why we should use SQL access on Hadoop, also giving an overview of IBM Big SQL and comparing it with Hive, Impala, and Hawq. This work provides a good insight on SQL-on-Hadoop tools and shows that mainly Hawq and Impala can provide good performance.

In [32] the authors present Impala giving an overview of its architecture and main components, also demonstrating its performance when compared against other popular SQL-on-Hadoop systems like Spark, Presto, and Hive, also presenting and comparing the compression ratio of popular combinations of file formats like Avro, Parquet and Text. They use the specific file format that performs best on each tool, revealing that Impala has faster response time executing queries in single and multi-user query execution.

Another benchmark of SQL-like Big Data technologies is presented in [36], which uses queries that involve table scans, aggregations and joins. When comparing Hive, Presto, Drill, and Spark they conclude that Presto has outstanding runtime on performance over other big data solutions and that SparkSQL has an edge for analytics/machine learning. In [40] characteristics like latency and ANSI SQL completeness, are used to evaluate Drill, Hive, Impala, and SparkSQL. It is highlighted that Hive has low maintenance and is simple to learn, but not suitable for real-time queries; Impala has lower query latency, but memory errors are very frequent. very frequent. In [41], the authors use a denormalized TPC-H schema testing it in a low cost cluster with Hive, Spark, Presto, and Drill. The results show that it's possible to achieve adequate query execution times on modest hardware. Although a comprehensive set of works in this field is already available, most of them are focus on extensive and sometimes confusing theoretical research, in some cases showing practical approaches and benchmarks in order to demonstrate tools performance.

In our work we select the state of the art Big Data processing tools, describing them very briefly, since its relatively easy to find information regarding these tools.

From our knowledge, the previously performed studies don't compare directly the set of state of the art tools what we chose to analyze. Our experimental evaluation involves Drill, Hawq, Hive, Presto, and Spark, also including Impala, which is very popular between users, but not compared in the previous works with all the tools referenced, maybe since it requires the deployment of distinct Hadoop distributions.

In our work we deployed all the chosen tools, using the most optimized file formats available for each tool, comparing Impala storing data using Parquet files and the remaining tools with the Stinger initiative ORC file format, performing a series of performance experiments, to conclude in which scenarios, they are suitable or ideal and those where is not recommended to use them.

3. SQL-on-Hadoop

Hadoop is one of the technologies that is immediately highlighted when dis-cussing Big Data. The Hadoop it has been a great tool for Big Data analysis, but it is still only accessible by a limited group of people, because of the huge efforts needed to learn its unique architecture. As previously referenced, SQL query processing for analytics over Hadoop data has recently gained significant traction, fueling the de-velopment of many SQL engines for Hadoop data, being Hive the first native Ha-doop querying system. We believe that SQL-on-Hadoop is an important concept in the context of making Big Data analysis accessible to more people and making data analysis easier and faster without the requirement of knowing the internals of the Hadoop ecosystems and Big Data concepts. In short, SQL-on-Hadoop systems is a class of analytical tools used to provide a SQL interface to Hadoop data framework elements, like HDFS, SQL or NoSQL databases and Hive.

3.1 Apache Drill

Drill is an open-source distributed system that supports data-intensive distributed applications for interactive ad hoc analysis of large-scale datasets. It was developed with the goal of providing low latency and faster interactive queries, supporting several data storage NoSQL databases like the Hadoop Distributed File System (HDFS), Hive or HBase and ANSI SQL to query data. To access this variety of data sources, Drill provides storage plugin interfaces to read from and write to data sources, defining a set of optimization rules to help with efficient and faster execution of Drill queries on a specific data source. Being simple to install on a cluster and using distributed cache to manage metadata, it can scale-up to a very large cluster with thousands of nodes. On top of all these characteristics, it offers connectivity and compatibility with BI tools like Tableau, Microstrategy, Qlikview and Tibco[36]. In short, Drill provides interactive query capabilities using its own SQL execution engine, enabling traditional Business Intelligence and analytics from different sources, suited for advanced analytics workflows, offering query response time that vary between milliseconds to minutes depending on the query complexity and the size of the dataset [43]. As a core component, it has the Drillbit service, reponsible for accepting requests from the client, processing the queries, and returning results to the client. Drillbits can be installed and executed on all nodes in a Hadoop cluster, forming a distributed cluster environment.

3.2 HAWQ

HAWQ (Hadoop With Query) was recently brought into the Apache Foundation from the well-established Pivotal HAWQ. It is a Hadoop native SQL query engine that combines the key technological advantages of Massive Parallel Processing (MPP) with the scalability and convenience of Hadoop, offering good performance. This parallel SQL query engine built on top of the HDFS, claims to be the world's fastest SQL engine on Hadoop [44], adopting a layered architecture and relies on HDFS for data replication and fault tolerance. HAWQ relies on both the PostgreSQL database and HDFS as its backend storage mechanism, providing support the full ANSI SQL syntax. Queries are executed via PostgreSQL client and then sent to HAWQ master, there they are parsed, optimized, fragmented and dispatched to HAWQ segments across the nodes for execution. During query execution YARN takes care of all resource management matters for overall performance and cluster stability.

3.3 Hive

Hive is a system that supports the processing and analysis of data stored in Hadoop, more specifically in Hadoop Distributed File System (HDFS). This tool its considered as the standard data warehouse storage system for Big Data and the first to support SQL-on-Hadoop, using an underlying framework such as MapReduce (or more recently, Tez) to process SQL-like statements. Hive gives structure to data and performs ad-hoc querying and analysis using HiveQL, a SQL-like query language. This query language has some limitations, since some SQL features not yet available such as update and delete queries, and also, several restrictions on the use of subqueries. Hive is frequently considered as a high-latency system, oriented to batch workloads instead of interactive querying, Hive has been target of constant development to improve its performance. Out of all these improvements we highlight the Stinger initiative that seeks to improve query performance and reduce latency.

Summarizing, Hive has a good interface for anyone from the relational database world, demanding low maintenance and being simple to learn. In the first distributions, . It is worth mentioning that using MapReduce is the main criticisms made to Hive, as the conversion to MapReduce jobs leads to higher query latency. Later versions of Hive can run on Tez, a tool that aims to enhance performance.

3.4 Impala

Impala is an open-source, state-of-the-art Massive Parallel Processing (MPP) SQL query engine designed for performance, real time, low latency and high concurrency processing with the objective of combining the SQL support and multi-user performance of a traditional analytical database with the scalability and flexibility of Apache Hadoop. This tool can use two storage systems, HDFS or HBase, being possible to query data in both. It can also be integrated with Business Intelligence tools like Tableau, Pentaho, Micro Strategy and Zoom Data. Running natively on Hadoop, it doesn't use Hadoop to run the queries, instead it uses a set of daemons installed on each DataNode tuned to optimize the local processing and avoid bottlenecks.

Impala relies on in-memory join implementations, meaning that if the amount of memory available ensures that data that is being processed by queries fits, in-memory processing can provide outstanding performances. On the other hand, if data processed cannot fit into memory [46], the execution can fail. In summary, Impala has a large advantage since processing is done in memory, reducing latency and Disk IO, especially in real-time and

ad-hoc queries, as long as the runtime is short enough that node failures during the query execution are unlikely and the data involved in the query fits in memory.

3.5 Presto

Presto is an open source distributed SQL query engine for running interactive analytical queries against data sources of all sizes, targeted at analysts who expect response times ranging from seconds to minutes. Using in-memory processing, instead of MapReduce, it avoids side steps, unnecessary I/O and latency, allowing faster response times. As a result, Facebook claims that Presto runs 10 times faster than Hive [36]. One of the main drawbacks is the limitation on the maximum amount of memory each query can have, so if a query requires a large amount of memory it will simply fail, similarly to what we described for Impala. In terms of architecture, in a nutshell, it runs on a cluster of machines that includes a coordinator, multiple workers and a client.

3.6 Spark

Spark is a highly distributed processing framework that provides an ease of use tool for efficient analytics on heterogeneous data. It was originally developed in 2009 in UC Berkeley's AMPLab, and open sourced in 2010 as an Apache project, running on top of Hadoop Distributed File System (HDFS). One of the key modules of Spark is Spark SQL, used for processing structured data with DataFrames (equivalent to relational database tables) that organize data in named columns. Spark provides full access and compatibility with Hive existing data and uses the concept of Resilient Distributed Datasets (RDD), which describes an immutable collection of objects that are partitioned and distributed across multiple nodes of a cluster, allowing parallel processing using a master/worker architecture, where a single coordinator (or master), manages all workers, executing tasks.

4. Experimental Enviroment and Data

After we gather the required clusters infrastructures we started to deploy the necessary software to run a stable Hadoop Environment, to support the installation and practical performance analysis of the previous analyzed tools. In this Chapter we describe the decisions made to deploy those tools and how we will evaluate their performance using the TPC benchmarks TPC-H and part of TPC-DS.

4.1 Hardware and software configuration

In order to run a distributed Hadoop environment in our Hadoop cluster, the 4 nodes should follow the master/worker architecture referenced on previous Chapters.

In short, the hardware configuration of each node includes one octa-core CPU 1.80GHz, 16 GB of RAM and one SATA disk with sizes ranging between 120 and 240 gigabytes, connected through gigabit Ethernet to achieve gigabit data rate, with 64-bit CentOS Linux 7 as the operating system. As seen frequently in the literature, SQL-on-Hadoop systems need significant amounts of RAM to process data, benefiting from the availability of a higher number of nodes available (e.g. cluster used in [1] with 21 nodes and 96 GB of RAM and [32] with 10 nodes and 48GB of RAM). Even though, as we see in [41], it's possible to perform this kind of experiences on low cost cluster. After we have our Hadoop cluster ready to use, it was time to install an Hadoop distribution, since our set of tools run on top of Hadoop. All the tools were deployed with on top of the Hortonworks HDP 2.5, except for Cloudera Impala. This tool required to Cloudera's distribution, we used CDH 5.

4.2 Storage Formats and data preparation

Data can be stored on Hadoop in numerous formats, such as Parquet, optimized row columnar (ORC) provide lightweight and fast access to compressed data with columnar layout, hence can significantly boost IO performance. As mentioned previously, the first tool approached was Hive, where we found that Stinger initiative improved tremendously its performance through the introduction of ORC file. All the other tools can support this file format as we seen previously, except Cloudera impala. We used Parquet file format in Impala which are the popular columnar formats that each system advertises [36].

5. Experimental Environment and results

This chapter presents all the results and analyses obtained performing the TPC-H and part TPC-DS benchmarks on the presented using 10, 50 and 100 GB datasets.

We run the all the 22 queries of TPC-H a subset of TPC-DS. The TPC-DS benchmark includes 99 queries, since it would consume a lot of time, we only performed a subset of the queries. The 99 queries cover various operational requirements and complexities (59 ad-hoc,41 reporting, 4 iterative OLAP and 23 data mining queries). In order to compare results of the two benchmarks, we performed some of the ad-hoc queries, since the 22 TPC-H are classified as ad-hoc. We performed the queries 5 times each, and attempt to eliminate time variations by cleaning file system cache before running each query. The final query execution time is the average of the 5 runs. We also the total time execution for all queries and also the speedup as lowest_tool_time/current_tool_time.

5.1 TPC-H Benchmark

On this benchmark, in overall, we observe that queries with complex joins and subqueries across several tables are the operations that are more demanding in terms of resources, requiring more processing time (e.g Q2, Q5, Q7, Q8, Q9, Q10, Q18 and Q21). Other costly operations include aggregates and arithmetic calculations present in several queries (e.g Q1, Q10, Q13, Q18, Q20, Q21) and large IN clauses against a series of at most eight constant values (e.g Q12, Q16, Q19 and Q22). As the scale factor (SF) grows, the used system starts to reach its limit, taking too long to process some queries or being, in some cases, unable to finish them, mainly for Presto as can be seen in Table 23. Analyzing the results of 10GB Scale factor, presented on Table 21, Hawq and Presto were the fastest tools with a speedup of 5.79 and 5.18, which means these tools more than 5 times faster than Spark. Although the overall execution time performed by Impala was longer than Hawq and Presto, it was able to queries ranging 1.2s to 36s being able to be the quickest in many queries like Q1, Q2, among others.

Table 21. Query Execution Time for 10 GB (in seconds)

	Drill	Hawq	Hive	Impala	Presto	Spark
Q1	8.83	5.88	17.80	2.87	4.49	26.73
Q2	12.69	6.13	23.31	2.66	10.98	36.21
Q3	8.18	6.50	20.49	11.83	7.61	32.72
Q4	6.77	5.34	34.39	10.62	6.05	60.18
Q5	13.21	10.08	23.33	19.50	10.6	34.54
Q6	4.61	3.49	10.84	1.23	2.66	18.82
Q7	20.36	5.94	51.26	22.95	12.59	46.51
Q8	13.10	4.60	23.32	31.62	10.23	31.84
Q9	21.36	11.06	36.77	35.77	16.16	52.97
Q10	11.12	7.48	32.30	6.28	6.78	44.02
Q11	3.77	3.92	16.00	1.15	4.17	23.42
Q 12	8.53	1.15	14.82	2.21	4.03	25.14
Q13	6.42	4.52	26.39	10.23	5.97	68.68
Q14	6.49	3.92	13.86	1.78	2.87	23.39
Q15	12.97	5.52	19.47	4.25	3.16	30.51
Q16	12.70	6.46	17.78	1.51	5.88	30.28
Q 17	9.20	15.62	16.22	1.68	8.41	23.85
Q18	32.77	14.83	42.21	12.18	14.40	86.33
Q 19	31.01	2.21	20.51	3.16	4.41	33.85
Q20	11.52	7.93	19.89	8.98	3.87	26.79
Q21	96.39	18.99	128.08	32.24	25.18	106.92
Q22	7.18	4.29	25.21	7.95	3.66	38.94
Total	359.18	155.86	634.25	232.65	174.16	902.64
Speedup	2.51	5.79	1.42	3.88	5.18	1

Regarding the 30 GB SF results presented on Table 22, the execution times took longer, but not linearly, meaning for 30GB, the execution time wasn´t 3x the execution times of 10GB.

For this SF presto surpassed Impala, being the fastest tool for this SF. Considering the total time required to run the 22 queries of the previous three fastest tools we see a decrease in the speedup of about 2.42, 2.66 and 1.97, meaning the overall performance of these tools improved.

For Q1, which computes eight aggregates: a count, four sums and three averages Drill was the slowest tool due to its performance taking 2s more than Spark. Also on Q2, besides performing multiple joins, it includes a LIKE string manipulation and a subquery that calculates minimum cost part supplier for a certain part, Drill was once again the slowest. On other queries, Drill could keep up with fastest tools performing similarly in some queries like Q3, Q4, Q5, Q11, Q13.

Table 22. Query Execution Time for 30 GB (in seconds)

	Drill	Hawq	Hive	Impala	Presto	Spark
Q1	30.93	15.2	25.78	13.21	13.10	28.82
Q2	58.12	13.32	36.86	11.36	13.43	38.47
Q3	20.97	24.47	29.72	52.50	16.72	36.75
Q4	18.96	21.01	85.66	51.49	12.31	125.66
Q5	28.79	26.67	45.00	68.46	27.59	49.29
Q6	7.70	8.83	10.24	5.57	3.47	22.61
Q7	42.67	13.66	76.40	78.68	34.18	82.52
Q8	17.19	12.21	32.56	12.11	26.68	41.86
Q9	27.68	25.22	182.58	164.2	37.75	126.68
Q10	27.72	23.26	62.17	17.72	14.36	63.17
Q11	4.22	8.88	23.68	6.88	9.14	52.15
Q 12	14.86	20.11	29.65	5.55	11.28	27.59
Q13	12.97	7.92	54.60	30.84	9.83	62.82
Q14	12.95	19.14	21.02	5.85	6.51	33.83
Q15	25.8	13.46	24.18	3.93	6.68	38.66
Q16	14.84	10.05	25.45	2.77	7.04	30.70
Q 17	11.21	44.22	22.92	3.88	21.12	30.50
Q18	52.38	38.73	130.77	41.24	44.55	97.68
Q 19	52.21	4.67	122.36	7.06	12.24	133.51
Q20	21.28	16.75	23.12	13.81	8.47	36.77
Q21	129.71	72.63	145.69	213.38	-	176.85
Q22	12.73	8.00	34.81	22.59	6.96	34.81
Total	516.18	375.78	1099.53	619.7	343.41	1194.85
Speedup	2.51	5.79	1.42	3.88	5.18	1

In the experiments when we reach 100GB SF, results presented on Table 23 the limit in terms of resources for our cluster is achieved, since we weren't able to run several queries, especially on Presto was unable to run 7 queries. Presto uses main memory in-memory processing, and since JOINS present in many of these queries process a considerable volume of data, as soon as the tool gets out of memory, the processing fails. Although Presto failed in many queries, it is worth mentioning that it was the fastest tool in almost all remaining queries (Q1, Q3, Q4, Q6, Q10, among others), leading us to believe that it in a cluster with more resources (RAM memory), it would be a suitable tool for interactive querying.

Regarding Drill, like in the previous kept perform similarly to the fastest tools in some queries (e.g Q4, Q5, Q16, Q18, Q19), meaning it's not a tool we should completely ignore although its surpassed largely by other tools in some queries. At this SF, we also notice that Spark does not present the slowest processing times (except in Q11 and Q16). Regarding HAWQ, the SF scability had a huge negative impact in its performance, having the longest times in several, on the other hand, it was the fastest one in the most complex query (Q21).

We notice that although Spark and Hive have not achieved the best results, they still managed to run all queries, leading us to acknowledge their robustness. In some queries, Hive present performances similar Impala and it even surpasses it in some cases like Q3, Q4, Q5, Q7, among others, although it was never able to surpass the times achieved by Presto.

As for Impala, similar to Presto, is very efficient if the processed data fits in the available memory. Otherwise, it applies a feature called "spill to disk", preventing out-of-memory errors. In those cases, data is written in disk and the query does not crash, although it greatly impairs the performance, which leads to extensive execution times that we see along the benchmark for Impala, the execution of Q9 and Q21 for the 100GB SF is a clear example of performance degradation, where we can observe execution time above 1000 seconds.

SQL clauses that require memory allocation that could activate the spilling mechanism include GROUP BY clause for columns with millions or billions of distinct values, JOIN operations over large tables and sorting operations performed by ORDER BY and GROUP BY clause.

Some other cases were already visible on the previous SF and keep degradation performance for the current SF e.g Q3, Q4, Q5, Q7, among other queries, than unlike smaller data volumes didn't provide almost instant query execution. We should notice we can configure presto to use "spill to disk", but unlike Impala it isn't activated by default. Since this feature improves robustness, but injures performance we didn't activate, in order to see how Presto deals with data growth.

Table 23. Query Execution Time for 100 GB (in seconds)

	Drill	Hawq	Hive	Impala	Presto	Spark
Q1	115.32	157.61	40.82	38.42	34.41	52.24
Q2	154.18	38.87	155.31	36.17	-	116.4
Q3	102.79	185.82	232.34	289.18	55.3	150.08
Q4	97.06	201.68	123.18	327.33	24.75	105.16
Q5	113.03	171.99	201.86	315.96	-	178.07
Q6	34.7	135.87	38.8	23.22	12.89	40.01
Q7	165.29	162.34	229.77	350.46	-	191.04
Q8	79.02	140.96	129.23	566.02	-	123.51
Q9	86.47	217.33	442.21	1070.66	-	420.5
Q10	98.55	150.81	259,13	60.52	46.04	143.69
Q11	16.35	28.65	31.63	10.64	29.78	44.09
Q 12	50.62	157.27	46.11	22.69	21.38	52.45
Q13	56.63	44.32	134.84	142.96	26.66	121.48
Q14	46.04	139.81	96.95	33.03	15.17	69.56
Q15	67.56	131.61	131.96	18.26	23.01	74.98
Q16	19.99	42.24	49.15	8.19	11.86	54.38
Q17	46.4	396.57	69.91	8.82	86.06	61.67
Q18	-	280.87	373.75	226.01	-	262.55
Q19	254.98	126.38	449.44	32.79	19.76	409.31
Q20	49.78	149.11	59.37	21.73	22.77	59.51
Q21	-	453.56	2870.82	1503.29	-	2869.76
Q22	17.2	41.84	96.19	62.82	14.37	76.44
Total	-	3555.51	6262.77	5169.17	-	5676.88
Speedup	-	1.76	1	1.21	-	1.10

5.2 TPC-DS ad hoc Queries

Interactive or ad hoc queries capture the dynamic nature of a Decision Support System. When we performed the TPC-H we observed that query performance is highly affected by heavy JOIN operations, arithmetical and aggregation functions, Regarding the 10GB dataset, results are presented on Table 24. TPC-DS Interactive Query Execution Time for 10 GB (in seconds)we can observe reasonable executions times in the overall.

Even though, like what we've seen in the TPC-H benchmark, Spark keeps being the slowest tool and surpassed by Hive in every query. The fastest tools for this SF were Impala and Hawq, where Impala surpasses Hawq in every except for Q43. Although Presto was always behind Impala and Hawq with significant differences, it was able to perform all the queries with quick response times, most of the queries ran under 10 seconds. We can see that these tools achieved nearly instant execution times, specially Impala reaching times below 1 second in some queries like Q12, Q42, Q55 and Q96, reaching overall speedup of 11.66 over the slowest tool Spark. The fastest tool Impala, have an above average executing time (comparing with other queries) executing Q82, the same happens in the second fastest tool Hawq. It's also the longest query ran on Hive and Spark, taking nearly 3x more than Drill, Hawq and Presto. These grater execution times can be explained since the query involves the 2 biggest fact tables Inventory and Store_sales, being sales the channel where more sales are performed, affecting execution times due to the volume of data processed.

Table 24. TPC-DS Interactive Query Execution Time for 10 GB (in seconds)

	Drill	Hawq	Hive	Impala	Presto	Spark
Q3	24.06	7.83	26.25	1.05	13.82	34.51
Q7	18.67	4.36	19.05	2.44	10.55	28.44
Q12	11.17	3.49	14.37	0.86	6.78	20.28
Q15	13.17	7.08	16.98	1.65	9.18	22.73
Q18	20.22	5.84	29.51	3.64	10.57	37.65
Q19	15.81	4.27	16.45	2.22	10.25	25.97
Q26	12.87	3.22	19.34	2.17	8.27	26.36
Q27	13.71	3.12	24.64	2.29	9.49	28.32
Q42	8.28	1.48	6.83	0.84	8.07	17.91
Q43	10.19	2.82	23.28	4.47	9.12	30.58
Q52	8.39	1.87	8.51	1.06	7.52	17.56
Q55	8.83	1.63	10.39	0.71	7.63	17.38
Q82	11.73	10.25	33.42	8.47	10.05	43.64
Q84	8.82	4.26	24.13	2.78	6.27	32.03
Q91	12.36	5.03	15.42	1.96	7.94	22.17
Q96	8.73	1.76	20.32	0.55	7.55	27.58
Total	207.01	51.74	308.89	37.16	143.61	433.11
Speedup	2.09	8.37	1.40	11.66	3.02	1

For the 30GB SF, one more time we observe that the execution times don't increase linarly linearly, we don't observe 3x execution times of 10GB. Similar to the 10GB sample, Table 25 shows that Hawq and Impala manage to be the fastest tools, but with this SF, Hawq was the quickest in the overall surpassing Impala. With the growth of SF Hawq surpassed the 10GB fastest tool Impala, managing to be fastest on Q7, Q18, Q19, Q27, Q43 (in this case nearly 50% faster) and Q84 for this SF. Hive kept surpassing Spark in every query having a 1.40 speedup regarding Spark. Presto was the third quickest tool, if we look into the overall query execution time, it is about 2x slower than HAWQ and Impala. However, still manages to have good performance in all queries. Like the previous SF, Q3 has the longest execution time, retrieving results in above average, comparing with the execution times of other queries times, the difference is more noticeable on Presto and Drill and specially on HAWQ. Q7 also keeps running on above average query execution time on Presto and specially on Drill (only 2 seconds faster than Hive) due to the heavy aggregates, ordering and grouping functions. Another case where this kind of operations affect performance can be observed when we perform Q18, having the longest running time time on Hive and the second slowest query on Drill, this query joins data from fact table Catalog_sales and 4 dimension tables, performing 7 average functions over Catalog_Sales columns. In the previous analyzed 10 GB SF, we identified Q82 as one of the more complex query in terms of processing, Impala was the fastest tool, followed by Presto, HAWQ and Drill with similar execution time. With the growth of data processed this difference became clearer, Presto performed best, beating Impala in nearly 9 seconds and Hawq by almost 14 seconds and Drill surpassed them, performing the query in 14.96 seconds.On Q43, Q42 and Q55, although with very few difference, Drill managed to surpass Presto, where within those 3 queries, except for Q43, Presto had almost the same performance as Hive.

Table 25. TPC-DS Interactive Query Execution Time for 30 GB (in seconds)

	Drill	Hawq	Hive	Impala	Presto	Spark
Q3	27.13	19.39	31.27	8.95	17.84	36.52
Q7	21.06	6.86	23.39	7.21	17.78	32.55
Q12	13.34	6.61	16.04	3.17	11.74	22.81
Q15	17.47	14.15	17.83	6.05	12.04	25.09
Q18	25.27	7.33	39.56	7.86	13.52	43.83
Q19	23.33	5.61	18.49	6.39	14.92	27.09
Q26	16.26	3.44	23.32	2.59	11.75	31.16
Q27	20.42	2.29	26.35	3.41	15.04	30.26
Q42	11.86	2.64	10.19	2.59	11.37	20.89
Q43	14.44	5.32	26.72	10.15	15.36	33.98

Q52	10.64	2.57	12.78	2.33	11.22	22.18
Q55	10.98	2.62	14.44	2.58	11.15	21.28
Q82	14.96	24.09	35.57	19.79	10.77	48.96
Q84	10.73	5.45	27.54	6.54	7.47	34.78
Q91	17.45	6.28	17.88	3.12	12.44	23.42
Q96	10.78	2.52	24.44	1.82	9.94	29.98
Total	267.02	83.63	365.81	94.55	204.35	484.78
Speedup	1.82	5.79	1.33	5.13	2.37	1

In the TPC-H benchmark, when we used 100 GB SF, our cluster resources started to reach its limit and consequently we couldn't run several queries, especially in Presto. Although in TPC-DS we observe that we can run every almost every query, for Presto we couldn't just run Q82 due to the causing IO intensive caused by complex joins, consequently as seen before, the amount data made Presto run out of memoy causing query execution failure. Cloudera Impala was again the fastest tool, like we saw on the 10GB SF, if we consider the total time it took to run all the queries, it leads with a huge time difference from the other engines. In the TPC-H we observe that Hive seemed to catch up the fastest tools as the data volume grows. On the interactive queries, when we reach 100GB SF, Hive manages to surpass HAWQ and Presto, and unlike what we saw on TPC-H, where Hive was never able to surpass Presto, now we can see that Hive to surpass Presto in several queries like Q3, Q7, Q15 among others. After comparing the results obtained we find that Hive is less suitable when querying in low data volume, due to the long coordination time of jobs, but if we look at the different SF, as the data volume grows, they seem to catch with the fastest tools, surpassing HAWQ in most of the queries. Evaluating the total time to execute all the queries, Hive as also surpassed Presto, being faster in ten out of sixteen queries. On the previous SF we saw that HAWQ was the fastest tool, followed by Impala. Now, similar to what we observed on THE TPC-H, when we reach this volume of data, it has a huge negative effect on HAWQ, in this case it performs as the slowest tool.

Table 26 TPC-DS Interactive Query Execution Time for 100 GB (in seconds)

	Drill	Hawq	Hive	Impala	Presto	Spark
Q3	35.21	56.17	36.47	29.28	49.21	76.59
Q7	65.81	58.79	35.62	32.65	45.67	98.29
Q12	25.65	34.11	19.37	8.74	17.77	95.16
Q15	39.67	66.72	23.74	19.62	32.76	37.12
Q18	45.25	53.34	65.18	20.17	44.18	71.41
Q19	61.36	74.39	26.97	15.77	47.05	32.97
Q26	45.33	67.61	33.29	7.09	33.01	40.73
Q27	62.58	65.24	28.25	11.14	63.42	38.74
Q42	21.51	59.84	15.84	5.37	43.67	27.88
Q43	25.39	58.43	41.92	46.49	48.94	56.19
Q52	19.42	56.16	16.92	5.67	42.56	27.12
Q55	18.02	56.32	17.09	4.25	38.37	29.65
Q82	54.96	69.53	137.58	101.34	-	121.06
Q84	20.11	16.28	42.72	30.82	13.62	46.81
Q91	22.78	16.23	19.75	11.51	20.47	50.41
Q96	21.68	59.55	38.78	5.91	33.59	49.04
Total (W/Q82)	529.77	799.18	461.91	254.48	574.2	762.71
Speedup	1.51	1	1.73	3.14	1.39	1.05

6. Conclusions

Big Data processing tools can deal with the massive quantity of data that exists nowadays, allowing users to perform ad-hoc querying, after performing these experiments we can consider that there is no one-size fits all

SQL-on-Hadoop tool. Presto, Impala and HAWQ started as the fastest tools, providing very fast response times, but with the data scale, things started to change to the point of regular query execution failures, here Hive proved its robustness, being able to perform all the queries. We can think on Hive as a standard, it was the first to support SQL-on-Hadoop and it is included in all Hadoop distributions. However, it is known to be only fitted to batch processing systems, our first results from the TPC-H benchmark led us to agree with this, even though we noticed that with the growth of data, Hive seemed to catch up with the fastest tools.

When we performed the TPC-DS, when we reached 100 GB we saw that Hive was capable to perform similarly and even surpass the fastest tools like Presto and HAWQ. Therefore, we have to acknowledge that Hive has incredible robustness, being able to perform all the queries, even if performance isn't the best, on some situations its preferable to take sometimes to get results then being unable to perform the queries like we see in Presto, even though in terms of real-time interactive analysis processing that requires very fast response times, we consider the best tools to be Impala, Presto and HAWQ, since we can see that these tools can provide the fastest response times.

Using in-memory processing it grants them great performance, we believe that the few resources of our cluster limited the performance of these tools and that's why we see them perform best when the data volume is lower and lose their edge when data scales. Regarding Impala, certain memory-intensive operations write temporary data to disk (known as spilling to disk) when Impala is close to exceeding its memory limit on a particular host. Thus, this feature improves reliability, the slowdown its significant. We referenced previously that Presto can use "spill to disk", we believe that activating it, it would be able to perform all queries, but with the referenced price.Both Impala and presto have great performance and are very popular between users, regarding query times Presto surpass Impala in some queries and Impala surpass Presto in others Presto seems to surpass it in some queries and vice-versa query response times, although, Presto has the advantage of being fully ANSI SQL compliant, and Impala only supports HiveQL.

For spark, we were surprised by the times obtained in the benchmark, since it was always the slowest tool, we lead us to believe that Spark its more suitable when the objective is not just to query data, but perform algorithmic analysis, statistics and Machine Learning. But so, would be Impala and HAWQ, although, Spark is the only tool suited for stream processing, making it the choice when dealing with continuous data streams. As future work we propose to extend the TPC-H and TPC-DS benchmark to higher scale factors to perform a deeper experimental evaluation of the big data querying tools analyzed. Also, since Hive is in constant development and was able to catch up with fastest tools, we should be alert for major current developments like the recent LLAP, and keep analyzing its performance. We believe Hive's performance can do more than batch processing, these constant developments are aiming to the application of Hive in real-time interactive analysis. Another interesting approach would be the analyses of the performance improvements with the scale of the cluster itself, adding more nodes and specially increasing RAM memory.

Appendix D – TPC-H Schema Creation and Load HiveQL Script

```
drop table if exists lineitem;
create external table lineitem
(L_ORDERKEY BIGINT,
 L_PARTKEY BIGINT,
 L_SUPPKEY BIGINT,
 L_LINENUMBER INT,
 L_QUANTITY DOUBLE,
 L_EXTENDEDPRICE DOUBLE,
 L_DISCOUNT DOUBLE,
 L_TAX DOUBLE,
 L_RETURNFLAG STRING,
 L_LINESTATUS STRING,
 L_SHIPDATE STRING,
 L_COMMITDATE STRING,
 L_RECEIPTDATE STRING,
 L_SHIPINSTRUCT STRING,
 L_SHIPMODE STRING,
 L_COMMENT STRING)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '|' STORED AS TEXTFILE
LOCATION '/hdfs_directory/lineitem';

drop table if exists part;
create external table part (P_PARTKEY BIGINT,
 P_NAME STRING,
 P_MFGR STRING,
 P_BRAND STRING,
 P_TYPE STRING,
 P_SIZE INT,
 P_CONTAINER STRING,
 P_RETAILPRICE DOUBLE,
 P_COMMENT STRING)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '|' STORED AS TEXTFILE
LOCATION '/hdfs_directory/part';

drop table if exists supplier;
create external table supplier (S_SUPPKEY BIGINT,
 S_NAME STRING,
 S_ADDRESS STRING,
 S_NATIONKEY BIGINT,
 S_PHONE STRING,
 S_ACCTBAL DOUBLE,
 S_COMMENT STRING)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '|' STORED AS TEXTFILE
LOCATION '/hdfs_directory/supplier/';

drop table if exists partsupp;
create external table partsupp (PS_PARTKEY BIGINT,
 PS_SUPPKEY BIGINT,
 PS_AVAILQTY INT,
```

```
 PS_SUPPLYCOST DOUBLE,
 PS_COMMENT STRING)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '|' STORED AS TEXTFILE
LOCATION'/hdfs_directory/partsupp';

drop table if exists nation;
create external table nation (N_NATIONKEY BIGINT,
 N_NAME STRING,
 N_REGIONKEY BIGINT,
 N_COMMENT STRING)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '|' STORED AS TEXTFILE
LOCATION '/hdfs_directory/nation';

drop table if exists region;
create external table region (R_REGIONKEY BIGINT,
 R_NAME STRING,
 R_COMMENT STRING)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '|' STORED AS TEXTFILE
LOCATION '/hdfs_directory/region';

drop table if exists customer;
create external table customer (C_CUSTKEY BIGINT,
 C_NAME STRING,
 C_ADDRESS STRING,
 C_NATIONKEY BIGINT,
 C_PHONE STRING,
 C_ACCTBAL DOUBLE,
 C_MKTSEGMENT STRING,
 C_COMMENT STRING)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '|' STORED AS TEXTFILE
LOCATION '/hdfs_directory/customer';

drop table if exists orders;
create external table orders (O_ORDERKEY BIGINT,
 O_CUSTKEY BIGINT,
 O_ORDERSTATUS STRING,
 O_TOTALPRICE DOUBLE,
 O_ORDERDATE STRING,
 O_ORDERPRIORITY STRING,
 O_CLERK STRING,
 O_SHIPPRIORITY INT,
 O_COMMENT STRING)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '|' STORED AS TEXTFILE
LOCATION '/hdfs_directory/orders';

LOAD DATA INPATH '/hdfs_directory/lineitem.tbl' OVERWRITE INTO TABLE lineitem;
LOAD DATA INPATH '/hdfs_directory/part.tbl' OVERWRITE INTO TABLE part;
LOAD DATA INPATH '/hdfs_directory/customer.tbl' OVERWRITE INTO TABLE customer;
LOAD DATA INPATH '/hdfs_directory/nation.tbl' OVERWRITE INTO TABLE nation;
LOAD DATA INPATH '/hdfs_directory/orders.tbl' OVERWRITE INTO TABLE orders;
LOAD DATA INPATH '/hdfs_directory/partsupp.tbl' OVERWRITE INTO TABLE partsupp;
LOAD DATA INPATH '/hdfs_directory/region.tbl' OVERWRITE INTO TABLE region;
LOAD DATA INPATH '/hdfs_directory/supplier.tbl' OVERWRITE INTO TABLE supplier;

create external table orc_lineitem like lineitem stored as ORC;
create external table orc_part like part stored as ORC;
create external table orc_customer like customer stored as ORC;
create external table orc_nation like nation stored as ORC;
create external table orc_orders like orders stored as ORC;
```

```
create external table orc_partsupp like partsupp stored as ORC;
create external table orc_region like region stored as ORC;
create external table orc_supplier like supplier stored as ORC;
create external table orc_lineitem like lineitem stored as ORC;
create external table orc_part like part stored as ORC;
create external table orc_customer like customer stored as ORC;
create external table orc_nation like nation stored as ORC;
create external table orc_orders like orders stored as ORC;
create external table orc_partsupp like partsupp stored as ORC;
create external table orc_region like region stored as ORC;
create external table orc_supplier like supplier stored as ORC;

insert overwrite orc_lineitem select * from lineitem;
insert overwrite orc_part select * from part;
insert overwrite orc_customer select * from customer;
insert overwrite orc_nation select * from nation ;
insert overwrite orc_orders select * from orders;
insert overwrite orc_partsupp select * from partsupp;
insert overwrite orc_region select * from region;
insert overwrite orc_supplier select * from supplier;

//PARQUET FORMAT TABLES

create external table prq_lineitem like lineitem stored as parquetfile;
create external table prq_part like part stored as parquetfile;
create external table prq_customer like customer stored as parquetfile;
create external table prq_nation like nation stored as parquetfile;
create external table prq_orders like orders stored as parquetfile;
create external table prq_partsupp like partsupp stored as parquetfile;
create external table prq_region like region stored as parquetfile;
create external table prq_supplier like supplier stored as parquetfile;

insert overwrite prq_lineitem select * from lineitem;
insert overwrite prq_part select * from part;
insert overwrite prq_customer select * from customer;
insert overwrite prq_nation select * from nation ;
insert overwrite prq_orders select * from orders;
insert overwrite prq_partsupp select * from partsupp;
insert overwrite prq_region select * from region;
insert overwrite prq_supplier select * from supplier;
```

Appendix E – TPC-DS Creation and Load HiveQL Script

```
create external table call_center(
      cc_call_center_sk         bigint
,    cc_call_center_id         string
,     cc_rec_start_date         string
,     cc_rec_end_date           string
,     cc_closed_date_sk         bigint
,     cc_open_date_sk           bigint
,     cc_name                   string
,     cc_class                  string
,     cc_employees              int
,     cc_sq_ft                  int
,     cc_hours                  string
,     cc_manager                string
,     cc_mkt_id                 int
,     cc_mkt_class              string
,     cc_mkt_desc               string
,     cc_market_manager         string
,     cc_division               int
,     cc_division_name          string
,     cc_company                int
,     cc_company_name           string
,     cc_street_number          string
,     cc_street_name            string
,     cc_street_type            string
,     cc_suite_number           string
,     cc_city                   string
,     cc_county                 string
,     cc_state                  string
,     cc_zip                    string
,     cc_country                string
,     cc_gmt_offset             double
,     cc_tax_percentage         double
)
row format delimited fields terminated by '|'
LOCATION  '/hdfs_directory/call_center';


create external table catalog_page(
      cp_catalog_page_sk        bigint
,     cp_catalog_page_id        string
,     cp_start_date_sk          bigint
,     cp_end_date_sk            bigint
,     cp_department             string
,     cp_catalog_number         int
,     cp_catalog_page_number    int
,     cp_description            string
,     cp_type                   string
)
row format delimited fields terminated by '|'
```

```
LOCATION  '/hdfs_directory/catalog_page';


create external table catalog_returns
(
    cr_returned_date_sk       bigint,
    cr_returned_time_sk       bigint,
    cr_item_sk                bigint,
    cr_refunded_customer_sk   bigint,
    cr_refunded_cdemo_sk      bigint,
    cr_refunded_hdemo_sk      bigint,
    cr_refunded_addr_sk       bigint,
    cr_returning_customer_sk  bigint,
    cr_returning_cdemo_sk     bigint,
    cr_returning_hdemo_sk     bigint,
    cr_returning_addr_sk      bigint,
    cr_call_center_sk         bigint,
    cr_catalog_page_sk        bigint,
    cr_ship_mode_sk           bigint,
    cr_warehouse_sk           bigint,
    cr_reason_sk              bigint,
    cr_order_number           bigint,
    cr_return_quantity        int,
    cr_return_amount          double,
    cr_return_tax             double,
    cr_return_amt_inc_tax     double,
    cr_fee                    double,
    cr_return_ship_cost       double,
    cr_refunded_cash          double,
    cr_reversed_charge        double,
    cr_store_credit           double,
    cr_net_loss               double
)
row format delimited fields terminated by '|'
LOCATION  '/hdfs_directory/catalog_returns';


create external table catalog_sales
(
    cs_sold_date_sk           bigint,
    cs_sold_time_sk           bigint,
    cs_ship_date_sk           bigint,
    cs_bill_customer_sk       bigint,
    cs_bill_cdemo_sk          bigint,
    cs_bill_hdemo_sk          bigint,
    cs_bill_addr_sk           bigint,
    cs_ship_customer_sk       bigint,
    cs_ship_cdemo_sk          bigint,
    cs_ship_hdemo_sk          bigint,
    cs_ship_addr_sk           bigint,
    cs_call_center_sk         bigint,
    cs_catalog_page_sk        bigint,
    cs_ship_mode_sk           bigint,
    cs_warehouse_sk           bigint,
    cs_item_sk                bigint,
    cs_promo_sk               bigint,
    cs_order_number           bigint,
    cs_quantity               int,
    cs_wholesale_cost         double,
```

```
    cs_list_price             double,
    cs_sales_price            double,
    cs_ext_discount_amt       double,
    cs_ext_sales_price        double,
    cs_ext_wholesale_cost     double,
    cs_ext_list_price         double,
    cs_ext_tax                double,
    cs_coupon_amt             double,
    cs_ext_ship_cost          double,
    cs_net_paid               double,
    cs_net_paid_inc_tax       double,
    cs_net_paid_inc_ship      double,
    cs_net_paid_inc_ship_tax  double,
    cs_net_profit             double
)
row format delimited fields terminated by '|'
LOCATION  '/hdfs_directory/catalog_sales';

create external table customer_address
(
    ca_address_sk             bigint,
    ca_address_id             string,
    ca_street_number          string,
    ca_street_name            string,
    ca_street_type            string,
    ca_suite_number           string,
    ca_city                   string,
    ca_county                 string,
    ca_state                  string,
    ca_zip                    string,
    ca_country                string,
    ca_gmt_offset             double,
    ca_location_type          string
)
row format delimited fields terminated by '|'
LOCATION  '/hdfs_directory/customer_address';

create external table customer_demographics
(
    cd_demo_sk                bigint,
    cd_gender                 string,
    cd_marital_status         string,
    cd_education_status       string,
    cd_purchase_estimate      int,
    cd_credit_rating          string,
    cd_dep_count              int,
    cd_dep_employed_count     int,
    cd_dep_college_count      int
)
row format delimited fields terminated by '|'
LOCATION  '/hdfs_directory/customer_demographics';



create external table customer
(
    c_customer_sk             bigint,
    c_customer_id             string,
    c_current_cdemo_sk        bigint,
```

```
    c_current_hdemo_sk        bigint,
    c_current_addr_sk         bigint,
    c_first_shipto_date_sk    bigint,
    c_first_sales_date_sk     bigint,
    c_salutation              string,
    c_first_name              string,
    c_last_name               string,
    c_preferred_cust_flag     string,
    c_birth_day               int,
    c_birth_month             int,
    c_birth_year              int,
    c_birth_country           string,
    c_login                   string,
    c_email_address           string,
    c_last_review_date        string
)
row format delimited fields terminated by '|'
LOCATION  '/hdfs_directory/customer';


create external table date_dim
(
    d_date_sk                 bigint,
    d_date_id                 string,
    d_date                    string,
    d_month_seq               int,
    d_week_seq                int,
    d_quarter_seq             int,
    d_year                    int,
    d_dow                     int,
    d_moy                     int,
    d_dom                     int,
    d_qoy                     int,
    d_fy_year                 int,
    d_fy_quarter_seq          int,
    d_fy_week_seq             int,
    d_day_name                string,
    d_quarter_name            string,
    d_holiday                 string,
    d_weekend                 string,
    d_following_holiday       string,
    d_first_dom               int,
    d_last_dom                int,
    d_same_day_ly             int,
    d_same_day_lq             int,
    d_current_day             string,
    d_current_week            string,
    d_current_month           string,
    d_current_quarter         string,
    d_current_year            string
)
row format delimited fields terminated by '|'
LOCATION  '/hdfs_directory/date_dim';

create external table household_demographics
(
    hd_demo_sk                bigint,
    hd_income_band_sk         bigint,
    hd_buy_potential          string,
```

```
    hd_dep_count              int,
    hd_vehicle_count          int
)
row format delimited fields terminated by '|'
LOCATION  '/hdfs_directory/household_demographics';

create external table income_band(
      ib_income_band_sk         bigint
,     ib_lower_bound            int
,     ib_upper_bound            int
)
row format delimited fields terminated by '|'
LOCATION  '/hdfs_directory/income_band';

create external table inventory
(
    inv_date_sk                 bigint,
    inv_item_sk                 bigint,
    inv_warehouse_sk            bigint,
    inv_quantity_on_hand        int
)
row format delimited fields terminated by '|'
LOCATION  '/hdfs_directory/inventory';



create external table item
(
    i_item_sk                 bigint,
    i_item_id                 string,
    i_rec_start_date          string,
    i_rec_end_date            string,
    i_item_desc               string,
    i_current_price           double,
    i_wholesale_cost          double,
    i_brand_id                int,
    i_brand                   string,
    i_class_id                int,
    i_class                   string,
    i_category_id             int,
    i_category                string,
    i_manufact_id             int,
    i_manufact                string,
    i_size                    string,
    i_formulation             string,
    i_color                   string,
    i_units                   string,
    i_container               string,
    i_manager_id              int,
    i_product_name            string
)
row format delimited fields terminated by '|'
LOCATION  '/hdfs_directory/item';


create external table promotion
(
    p_promo_sk                bigint,
    p_promo_id                string,
```

```
    p_start_date_sk           bigint,
    p_end_date_sk             bigint,
    p_item_sk                 bigint,
    p_cost                    double,
    p_response_target         int,
    p_promo_name              string,
    p_channel_dmail           string,
    p_channel_email           string,
    p_channel_catalog         string,
    p_channel_tv              string,
    p_channel_radio           string,
    p_channel_press           string,
    p_channel_event           string,
    p_channel_demo            string,
    p_channel_details         string,
    p_purpose                 string,
    p_discount_active         string
)
row format delimited fields terminated by '|'
LOCATION  '/hdfs_directory/promotion';

drop table if exists reason;

create external table reason(
      r_reason_sk               bigint
,     r_reason_id               string
,     r_reason_desc             string
)
row format delimited fields terminated by '|'
LOCATION  '/hdfs_directory/reason';


drop table if exists ship_mode;

create external table ship_mode(
      sm_ship_mode_sk           bigint
,     sm_ship_mode_id           string
,     sm_type                   string
,     sm_code                   string
,     sm_carrier                string
,     sm_contract               string
)
row format delimited fields terminated by '|'
LOCATION  '/hdfs_directory/ship_mode';


create external table store_returns
(
    sr_returned_date_sk       bigint,
    sr_return_time_sk         bigint,
    sr_item_sk                bigint,
    sr_customer_sk            bigint,
    sr_cdemo_sk               bigint,
    sr_hdemo_sk               bigint,
    sr_addr_sk                bigint,
    sr_store_sk               bigint,
    sr_reason_sk              bigint,
    sr_ticket_number          bigint,
    sr_return_quantity        int,
```

```
    sr_return_amt             double,
    sr_return_tax             double,
    sr_return_amt_inc_tax     double,
    sr_fee                    double,
    sr_return_ship_cost       double,
    sr_refunded_cash          double,
    sr_reversed_charge        double,
    sr_store_credit           double,
    sr_net_loss               double
)
row format delimited fields terminated by '|'
LOCATION  '/hdfs_directory/store_returns';

create external table store_sales
(
    ss_sold_date_sk           bigint,
    ss_sold_time_sk           bigint,
    ss_item_sk                bigint,
    ss_customer_sk            bigint,
    ss_cdemo_sk               bigint,
    ss_hdemo_sk               bigint,
    ss_addr_sk                bigint,
    ss_store_sk               bigint,
    ss_promo_sk               bigint,
    ss_ticket_number          bigint,
    ss_quantity               int,
    ss_wholesale_cost         double,
    ss_list_price             double,
    ss_sales_price            double,
    ss_ext_discount_amt       double,
    ss_ext_sales_price        double,
    ss_ext_wholesale_cost     double,
    ss_ext_list_price         double,
    ss_ext_tax                double,
    ss_coupon_amt             double,
    ss_net_paid               double,
    ss_net_paid_inc_tax       double,
    ss_net_profit             double
)
row format delimited fields terminated by '|'
LOCATION  '/hdfs_directory/store_sales';

create external table store
(
    s_store_sk                bigint,
    s_store_id                string,
    s_rec_start_date          string,
    s_rec_end_date            string,
    s_closed_date_sk          bigint,
    s_store_name              string,
    s_number_employees        int,
    s_floor_space             int,
    s_hours                   string,
    s_manager                 string,
    s_market_id               int,
    s_geography_class         string,
    s_market_desc             string,
    s_market_manager          string,
    s_division_id             int,
```

```
    s_division_name           string,
    s_company_id              int,
    s_company_name            string,
    s_street_number           string,
    s_street_name             string,
    s_street_type             string,
    s_suite_number            string,
    s_city                    string,
    s_county                  string,
    s_state                   string,
    s_zip                     string,
    s_country                 string,
    s_gmt_offset              double,
    s_tax_precentage          double
)
row format delimited fields terminated by '|'
LOCATION  '/hdfs_directory/store';


drop table if exists time_dim;

create external table time_dim
(
    t_time_sk                 bigint,
    t_time_id                 string,
    t_time                    int,
    t_hour                    int,
    t_minute                  int,
    t_second                  int,
    t_am_pm                   string,
    t_shift                   string,
    t_sub_shift               string,
    t_meal_time               string
)
row format delimited fields terminated by '|'
LOCATION  '/hdfs_directory/time_dim';

create external table warehouse(
      w_warehouse_sk            bigint
,     w_warehouse_id            string
,     w_warehouse_name          string
,     w_warehouse_sq_ft         int
,     w_street_number           string
,     w_street_name             string
,     w_street_type             string
,     w_suite_number            string
,     w_city                    string
,     w_county                  string
,     w_state                   string
,     w_zip                     string
,     w_country                 string
,     w_gmt_offset              double
)
row format delimited fields terminated by '|'
LOCATION  '/hdfs_directory/warehouse';


drop table if exists web_page;
```

```
create external table web_page(
      wp_web_page_sk            bigint
,     wp_web_page_id            string
,     wp_rec_start_date        string
,     wp_rec_end_date          string
,     wp_creation_date_sk       bigint
,     wp_access_date_sk         bigint
,     wp_autogen_flag           string
,     wp_customer_sk            bigint
,     wp_url                    string
,     wp_type                   string
,     wp_char_count             int
,     wp_link_count             int
,     wp_image_count            int
,     wp_max_ad_count           int
)
row format delimited fields terminated by '|'
LOCATION  '/hdfs_directory/web_page';

create external table web_returns
(
    wr_returned_date_sk       bigint,
    wr_returned_time_sk       bigint,
    wr_item_sk                bigint,
    wr_refunded_customer_sk   bigint,
    wr_refunded_cdemo_sk      bigint,
    wr_refunded_hdemo_sk      bigint,
    wr_refunded_addr_sk       bigint,
    wr_returning_customer_sk  bigint,
    wr_returning_cdemo_sk     bigint,
    wr_returning_hdemo_sk     bigint,
    wr_returning_addr_sk      bigint,
    wr_web_page_sk            bigint,
    wr_reason_sk              bigint,
    wr_order_number           bigint,
    wr_return_quantity        int,
    wr_return_amt             double,
    wr_return_tax             double,
    wr_return_amt_inc_tax     double,
    wr_fee                    double,
    wr_return_ship_cost       double,
    wr_refunded_cash          double,
    wr_reversed_charge        double,
    wr_account_credit         double,
    wr_net_loss               double
)
row format delimited fields terminated by '|'
LOCATION  '/hdfs_directory/web_returns';


create external table web_sales
(
    ws_sold_date_sk           bigint,
    ws_sold_time_sk           bigint,
    ws_ship_date_sk           bigint,
    ws_item_sk                bigint,
    ws_bill_customer_sk       bigint,
    ws_bill_cdemo_sk          bigint,
    ws_bill_hdemo_sk          bigint,
```

```
    ws_bill_addr_sk           bigint,
    ws_ship_customer_sk       bigint,
    ws_ship_cdemo_sk          bigint,
    ws_ship_hdemo_sk          bigint,
    ws_ship_addr_sk           bigint,
    ws_web_page_sk            bigint,
    ws_web_site_sk            bigint,
    ws_ship_mode_sk           bigint,
    ws_warehouse_sk           bigint,
    ws_promo_sk               bigint,
    ws_order_number           bigint,
    ws_quantity               int,
    ws_wholesale_cost         double,
    ws_list_price             double,
    ws_sales_price            double,
    ws_ext_discount_amt       double,
    ws_ext_sales_price        double,
    ws_ext_wholesale_cost     double,
    ws_ext_list_price         double,
    ws_ext_tax                double,
    ws_coupon_amt             double,
    ws_ext_ship_cost          double,
    ws_net_paid               double,
    ws_net_paid_inc_tax       double,
    ws_net_paid_inc_ship      double,
    ws_net_paid_inc_ship_tax  double,
    ws_net_profit             double
)
row format delimited fields terminated by '|'
LOCATION  '/hdfs_directory/web_sales';

create external table web_site
(
    web_site_sk           bigint,
    web_site_id           string,
    web_rec_start_date    string,
    web_rec_end_date      string,
    web_name              string,
    web_open_date_sk      bigint,
    web_close_date_sk     bigint,
    web_class             string,
    web_manager           string,
    web_mkt_id            int,
    web_mkt_class         string,
    web_mkt_desc          string,
    web_market_manager    string,
    web_company_id        int,
    web_company_name      string,
    web_street_number     string,
    web_street_name       string,
    web_street_type       string,
    web_suite_number      string,
    web_city              string,
    web_county            string,
    web_state             string,
    web_zip               string,
    web_country           string,
    web_gmt_offset        double,
    web_tax_percentage    double
```

```
)
row format delimited fields terminated by '|'
LOCATION  '/hdfs_directory/web_site';

LOAD DATA INPATH '/hdfs_directory//call_center.dat' OVERWRITE INTO TABLE
call_center;
LOAD DATA INPATH '/hdfs_directory//catalog_page.dat' OVERWRITE INTO TABLE
catalog_page;
LOAD DATA INPATH '/hdfs_directory//catalog_returns.dat' OVERWRITE INTO TABLE
catalog_returns;
LOAD DATA INPATH '/hdfs_directory//catalog_sales.dat' OVERWRITE INTO TABLE
catalog_sales;
LOAD DATA INPATH '/hdfs_directory//customer.dat' OVERWRITE INTO TABLE customer;
LOAD DATA INPATH '/hdfs_directory//customer_address.dat' OVERWRITE INTO TABLE
customer_address;
LOAD DATA INPATH '/hdfs_directory//customer_demographics.dat' OVERWRITE INTO TABLE
customer_demographics;
LOAD DATA INPATH '/hdfs_directory//date_dim.dat' OVERWRITE INTO TABLE date_dim;
LOAD DATA INPATH '/hdfs_directory//household_demographics.dat' OVERWRITE INTO
TABLE household_demographics;
LOAD DATA INPATH '/hdfs_directory//income_band.dat' OVERWRITE INTO TABLE
income_band;
LOAD DATA INPATH '/hdfs_directory//inventory.dat' OVERWRITE INTO TABLE inventory;
LOAD DATA INPATH '/hdfs_directory//item.dat' OVERWRITE INTO TABLE item;
LOAD DATA INPATH '/hdfs_directory//promotion.dat' OVERWRITE INTO TABLE promotion;
LOAD DATA INPATH '/hdfs_directory//reason.dat' OVERWRITE INTO TABLE reason;
LOAD DATA INPATH '/hdfs_directory//ship_mode.dat' OVERWRITE INTO TABLE ship_mode;
LOAD DATA INPATH '/hdfs_directory//store.dat' OVERWRITE INTO TABLE store;
LOAD DATA INPATH '/hdfs_directory//store_returns.dat' OVERWRITE INTO TABLE
store_returns;
LOAD DATA INPATH '/hdfs_directory//store_sales.dat' OVERWRITE INTO TABLE
store_sales;
LOAD DATA INPATH '/hdfs_directory//time_dim.dat' OVERWRITE INTO TABLE time_dim;
LOAD DATA INPATH '/hdfs_directory//warehouse.dat' OVERWRITE INTO TABLE warehouse;
LOAD DATA INPATH '/hdfs_directory//web_page.dat' OVERWRITE INTO TABLE web_page;
LOAD DATA INPATH '/hdfs_directory//web_returns.dat' OVERWRITE INTO TABLE
web_returns;
LOAD DATA INPATH '/hdfs_directory//web_site.dat' OVERWRITE INTO TABLE web_site;
LOAD DATA INPATH '/hdfs_directory//web_sales.dat' OVERWRITE INTO TABLE web_sales;

create table pq_call_center like call_center stored as parquetfile;
create table pq_catalog_page like catalog_page stored as parquetfile;
create table pq_catalog_returns like catalog_returns stored as parquetfile;
create table pq_catalog_sales like catalog_sales stored as parquetfile;
create table pq_customer like customer stored as parquetfile;
create table pq_customer_address like customer_address stored as parquetfile;
create table pq_customer_demographics like customer_demographics stored as
parquetfile;
create table pq_date_dim like date_dim stored as parquetfile;
create table pq_household_demographics like household_demographics stored as
parquetfile;
create table pq_income_band like income_band stored as parquetfile;
create table pq_inventory like inventory stored as parquetfile;
create table pq_item like item stored as parquetfile;
create table pq_promotion like promotion stored as parquetfile;
create table pq_reason like reason stored as parquetfile;
create table pq_ship_mode like ship_mode stored as parquetfile;
create table pq_store like store stored as parquetfile;
create table pq_store_returns like store_returns stored as parquetfile;
```

```
create table pq_store_sales like store_sales  stored as parquetfile;
create table pq_time_dim like time_dim stored as parquetfile;
create table pq_warehouse like warehouse stored as parquetfile;
create table pq_web_page like web_page stored as parquetfile;
create table pq_web_returns like web_returns stored as parquetfile;
create table pq_web_sales like web_sales stored as parquetfile;
create table pq_web_site like web_site stored as parquetfile;




insert overwrite pq_call_center select * from call_center;
insert overwrite pq_catalog_page select * from catalog_page;
insert overwrite pq_catalog_returns select * from catalog_returns;
insert overwrite pq_catalog_sales select * from catalog_sales;
insert overwrite pq_customer select * from customer;
insert overwrite pq_customer_address select * from customer_address;
insert overwrite pq_customer_demographics select * from customer_demographics;
insert overwrite pq_date_dim select * from date_dim;
insert overwrite pq_household_demographics select * from household_demographics;
insert overwrite pq_income_band select * from income_band;
insert overwrite pq_inventory select * from inventory;
insert overwrite pq_item select * from item;
insert overwrite pq_promotion select * from promotion;
insert overwrite pq_reason select * from reason;
insert overwrite pq_ship_mode select * from ship_mode;
insert overwrite pq_store select * from store;
insert overwrite pq_store_returns select * from store_returns;
insert overwrite pq_store_sales select * from store_sales;
insert overwrite pq_time_dim select * from time_dim ;
insert overwrite pq_warehouse select * from warehouse;
insert overwrite pq_web_page select * from web_page;
insert overwrite pq_web_returns select * from web_returns;
insert overwrite pq_web_sales select * from web_sales;
insert overwrite pq_web_site select * from web_site;

create table orc_call_center like call_center stored as parquetfile;
create table orc_catalog_page like catalog_page stored as parquetfile;
create table orc_catalog_returns like catalog_returns stored as parquetfile;
create table orc_catalog_sales like catalog_sales stored as parquetfile;
create table orc_customer like customer stored as parquetfile;
create table orc_customer_address like customer_address stored as parquetfile;
create table orc_customer_demographics like customer_demographics stored as
parquetfile;
create table orc_date_dim like date_dim stored as parquetfile;
create table orc_household_demographics like household_demographics stored as
parquetfile;
create table orc_income_band like income_band stored as parquetfile;
create table orc_inventory like inventory stored as parquetfile;
create table orc_item like item stored as parquetfile;
create table orc_promotion like promotion stored as parquetfile;
create table orc_reason like reason stored as parquetfile;
create table orc_ship_mode like ship_mode stored as parquetfile;
create table orc_store like store stored as parquetfile;
create table orc_store_returns like store_returns stored as parquetfile;
create table orc_store_sales like store_sales  stored as parquetfile;
create table orc_time_dim like time_dim stored as parquetfile;
create table orc_warehouse like warehouse stored as parquetfile;
create table orc_web_page like web_page stored as parquetfile;
```

```
create table orc_web_returns like web_returns stored as parquetfile;
create table orc_web_sales like web_sales stored as parquetfile;
create table orc_web_site like web_site stored as parquetfile;



insert overwrite orc_call_center select * from call_center;
insert overwrite orc_catalog_page select * from catalog_page;
insert overwrite orc_catalog_returns select * from catalog_returns;
insert overwrite orc_catalog_sales select * from catalog_sales;
insert overwrite orc_customer select * from customer;
insert overwrite orc_customer_address select * from customer_address;
insert overwrite orc_customer_demographics select * from customer_demographics;
insert overwrite orc_date_dim select * from date_dim;
insert overwrite orc_household_demographics select * from household_demographics;
insert overwrite orc_income_band select * from income_band;
insert overwrite orc_inventory select * from inventory;
insert overwrite orc_item select * from item;
insert overwrite orc_promotion select * from promotion;
insert overwrite orc_reason select * from reason;
insert overwrite orc_ship_mode select * from ship_mode;
insert overwrite orc_store select * from store;
insert overwrite orc_store_returns select * from store_returns;
insert overwrite orc_store_sales select * from store_sales;
insert overwrite orc_time_dim select * from time_dim ;
insert overwrite orc_warehouse select * from warehouse;
insert overwrite orc_web_page select * from web_page;
insert overwrite orc_web_returns select * from web_returns;
insert overwrite pq_web_sales select * from web_sales;
insert overwrite pq_web_site select * from web_site;
```

Appendix F – TPC-H Benchmark Queries

QUERY 1: Pricing Summary Report

The Pricing Summary Report Query provides a summary pricing report for all line items shipped as of a given date.The query lists totals for extended price, discounted extended price, discounted extended price plus tax, average quantity, average extended price, and average discount. These aggregates are grouped by RETURNFLAG and LINESTATUS, and listed in ascending order of RETURNFLAG and LINESTATUS. A count of the number of line items in each group is included:

```
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*(1L_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(*)
ASCOUNT_ORDER
FROM LINEITEM
WHERE L_SHIPDATE <= dateadd(dd, -90, cast('1998-12-01' as datetime))
GROUP BY L_RETURNFLAG, L_LINESTATUS
ORDER BY L_RETURNFLAG,L_LINESTATUS
```

QUERY 2: Minimum Cost Supplier

This query will find, in a give region for each part of a certain type and size, the supplier that can supply it at the lowest cost. If multiple suppliers in that region offer the same lowest price for the part, the query will list the parts form the suppliers with the 100 highest account balances

```
SELECT TOP 100 S_ACCTBAL, S_NAME, N_NAME, P_PARTKEY, P_MFGR, S_ADDRESS, S_PHONE,
S_COMMENT
FROM PART, SUPPLIER, PARTSUPP, NATION, REGION
WHERE P_PARTKEY = PS_PARTKEY AND S_SUPPKEY = PS_SUPPKEY AND P_SIZE = 15 AND P_TYPE
LIKE '%%BRASS' AND S_NATIONKEY = N_NATIONKEY AND N_REGIONKEY = R_REGIONKEY AND
R_NAME = 'EUROPE' AND PS_SUPPLYCOST = (SELECT MIN(PS_SUPPLYCOST) FROM PARTSUPP,
SUPPLIER, NATION, REGION WHERE P_PARTKEY = PS_PARTKEY AND S_SUPPKEY = PS_SUPPKEY
AND S_NATIONKEY = N_NATIONKEY AND N_REGIONKEY = R_REGIONKEY AND R_NAME = 'EUROPE')
ORDER BY S_ACCTBAL DESC, N_NAME, S_NAME, P_PARTKEY
```

QUERY 3: Shipping Priority

This query will discover the shipping priority and potential revenue, defined as the sum of extended price of the orders having the largest revenue among those that had not been shipped as of a given date. If more than 10 unshipped orders exist, only the 10 orders with the largest revenue are listed.

```
SELECT TOP 10 L_ORDERKEY, SUM(L_EXTENDEDPRICE*(1-L_DISCOUNT)) AS REVENUE,
O_ORDERDATE, O_SHIPPRIORITY
FROM CUSTOMER, ORDERS, LINEITEM
WHERE C_MKTSEGMENT = 'BUILDING' AND C_CUSTKEY = O_CUSTKEY AND L_ORDERKEY =
O_ORDERKEY ANDO_ORDERDATE < '1995-03-15' AND L_SHIPDATE > '1995-03-15'
GROUP BY L_ORDERKEY, O_ORDERDATE, O_SHIPPRIORITY
ORDER BY REVENUE DESC, O_ORDERDATE
```

QUERY 4: Order Priority Checking:

This query will count the number of orders that were ordered in a given quarter of a given year in wich at least one line item as received later than its committed date.

```
SELECT O_ORDERPRIORITY, COUNT(*) AS ORDER_COUNT FROM ORDERS
WHERE O_ORDERDATE >= '1993-07-01'
AND O_ORDERDATE < dateadd(mm,3, cast('1993-07-01' as datetime))
AND EXISTS (SELECT * FROM LINEITEM WHERE L_ORDERKEY = O_ORDERKEY
AND L_COMMITDATE < L_RECEIPTDATE)
GROUP BY O_ORDERPRIORITY
ORDER BY O_ORDERPRIORITY
```

QUERY 5: Local Supplier Volume:

This query will list, for each country in a region, the revenue volume that resulted from line item transactions in wich the customer ordering parts and the supplier filling them were both in the same country. The query only considers parts ordered a certain year.

```
SELECT N_NAME, SUM(L_EXTENDEDPRICE*(1-L_DISCOUNT)) AS REVENUE
FROM CUSTOMER, ORDERS, LINEITEM, SUPPLIER, NATION, REGION
WHERE C_CUSTKEY = O_CUSTKEY AND L_ORDERKEY = O_ORDERKEY AND L_SUPPKEY = S_SUPPKEY
AND C_NATIONKEY = S_NATIONKEY AND S_NATIONKEY = N_NATIONKEY AND N_REGIONKEY =
R_REGIONKEY
AND R_NAME = 'ASIA' AND O_ORDERDATE >= '1994-01-01'
AND O_ORDERDATE < DATEADD(YY, 1, cast('1994-01-01' as datetime))
GROUP BY N_NAME
ORDER BY REVENUE DESC
```

QUERY 6: Forecasting Revenue Change

This query will quantify the amount of revenue increase that would have resulted from eliminating certain company-wide discounts in a given percentage range in a given year:

```
SELECT SUM(L_EXTENDEDPRICE*L_DISCOUNT) AS REVENUE
FROM LINEITEM
WHERE L_SHIPDATE >= '1994-01-01' AND L_SHIPDATE < dateadd(yy, 1, cast('1994-01-01'
as datetime))AND L_DISCOUNT BETWEEN .06 - 0.01 AND .06 + 0.01 AND L_QUANTITY < 24
```

QUERY 7: Volume Shipping

This query will determine the value of goods shipped between certain countries to help in the renegotiation of shipping contracts.

```
SELECT SUPP_NATION, CUST_NATION, L_YEAR, SUM(VOLUME) AS REVENUE
FROM ( SELECT N1.N_NAME AS SUPP_NATION, N2.N_NAME AS CUST_NATION, datepart(yy,
L_SHIPDATE) AS L_YEAR,
L_EXTENDEDPRICE*(1-L_DISCOUNT) AS VOLUME
FROM SUPPLIER, LINEITEM, ORDERS, CUSTOMER, NATION N1, NATION N2
WHERE S_SUPPKEY = L_SUPPKEY AND O_ORDERKEY = L_ORDERKEY AND C_CUSTKEY = O_CUSTKEY
AND S_NATIONKEY = N1.N_NATIONKEY AND C_NATIONKEY = N2.N_NATIONKEY AND((N1.N_NAME =
'FRANCE' AND N2.N_NAME = 'GERMANY') OR N1.N_NAME = 'GERMANY' AND N2.N_NAME =
'FRANCE')) AND
L_SHIPDATE BETWEEN '1995-01-01' AND '1996-12-31') AS SHIPPING
GROUP BY SUPP_NATION, CUST_NATION, L_YEAR
ORDER BY SUPP_NATION, CUST_NATION, L_YEAR
```

QUERY 8: National Market Share

This query will determine how the market share of a given country within a given region has changed over the two years for a given part type.

```
SELECT O_YEAR, SUM(CASE WHEN NATION= 'BRAZIL' THEN VOLUME ELSE 0 END)/SUM(VOLUME)
AS MKT_SHARE FROM (SELECT datepart(yy,O_ORDERDATE) AS O_YEAR, L_EXTENDEDPRICE*(1-
L_DISCOUNT) AS VOLUME, N2.N_NAME AS NATION
FROM PART, SUPPLIER, LINEITEM, ORDERS, CUSTOMER, NATION N1, NATION N2, REGION
WHERE P_PARTKEY = L_PARTKEY AND S_SUPPKEY = L_SUPPKEY AND L_ORDERKEY = O_ORDERKEY
AND O_CUSTKEY = C_CUSTKEY AND C_NATIONKEY = N1.N_NATIONKEY AND
N1.N_REGIONKEY = R_REGIONKEY AND R_NAME = 'AMERICA' AND S_NATIONKEY =
N2.N_NATIONKEY
AND O_ORDERDATE BETWEEN '1995-01-01' AND '1996-12-31' AND P_TYPE= 'ECONOMY
ANODIZED STEEL') AS ALL_NATIONS
GROUP BY O_YEAR
ORDER BY O_YEAR
```

QUERY 9: Product Type Profit Measure

This query determines how much profit is made on a given line of parts, broken out by supplier country and year.

```
SELECT NATION, O_YEAR, SUM(AMOUNT) AS SUM_PROFIT
FROM (SELECT N_NAME AS NATION, datepart(yy, O_ORDERDATE) AS O_YEAR,
L_EXTENDEDPRICE*(1-L_DISCOUNT)-PS_SUPPLYCOST*L_QUANTITY AS AMOUNT
FROM PART, SUPPLIER, LINEITEM, PARTSUPP, ORDERS, NATION
WHERE S_SUPPKEY = L_SUPPKEY AND PS_SUPPKEY= L_SUPPKEY AND PS_PARTKEY = L_PARTKEY
AND P_PARTKEY= L_PARTKEY AND O_ORDERKEY = L_ORDERKEY AND S_NATIONKEY = N_NATIONKEY
AND P_NAME LIKE '%%green%%') AS PROFIT
GROUP BY NATION, O_YEAR
ORDER BY NATION, O_YEAR DESC
```

QUERY 10: Returned Item Reporting

This query identifies customers who might be having problems with the parts that are shipped to them.

```
SELECT TOP 20 C_CUSTKEY, C_NAME, SUM(L_EXTENDEDPRICE*(1-L_DISCOUNT)) AS REVENUE,
C_ACCTBAL,N_NAME, C_ADDRESS, C_PHONE, C_COMMENT FROM CUSTOMER, ORDERS, LINEITEM,
NATION
WHERE C_CUSTKEY = O_CUSTKEY AND L_ORDERKEY = O_ORDERKEY AND O_ORDERDATE>= '1993-
10-01' AND O_ORDERDATE < dateadd(mm, 3, cast('1993-10-01' as datetime)) AND
L_RETURNFLAG = 'R' AND C_NATIONKEY = N_NATIONKEY GROUP BY C_CUSTKEY, C_NAME,
C_ACCTBAL, C_PHONE, N_NAME, C_ADDRESS, C_COMMENT ORDER BY REVENUE DESC
```

QUERY 11: Important Stock Identification

This query finds the most important subset of supplier's stock in a given country;

```
SELECT PS_PARTKEY, SUM(PS_SUPPLYCOST*PS_AVAILQTY) AS VALUE
FROM PARTSUPP, SUPPLIER, NATION
WHERE PS_SUPPKEY = S_SUPPKEY AND S_NATIONKEY = N_NATIONKEY AND N_NAME = 'GERMANY'
GROUP BY PS_PARTKEY
HAVING SUM(PS_SUPPLYCOST*PS_AVAILQTY) > (SELECT SUM(PS_SUPPLYCOST*PS_AVAILQTY) *
0.0001000000
FROM PARTSUPP, SUPPLIER, NATION
WHERE PS_SUPPKEY = S_SUPPKEY AND S_NATIONKEY = N_NATIONKEY AND N_NAME
ORDER BY VALUE DESC
```

QUERY 12: Shipping Modes and Order Priority

This query determines whether selecting less expensive modes of shipping is negatively affecting the critical-priority orders by causing more parts to be received by customers after the committed date;

```
SELECT L_SHIPMODE, SUM(CASE WHEN O_ORDERPRIORITY = '1-URGENT' OR O_ORDERPRIORITY =
'2-HIGH' THEN 1 ELSE 0 END) AS HIGH_LINE_COUNT,SUM(CASE WHEN O_ORDERPRIORITY <>
'1-URGENT' AND O_ORDERPRIORITY <> '2-HIGH' THEN 1 ELSE 0 END ) AS LOW_LINE_COUNT
FROM ORDERS, LINEITEM
WHERE O_ORDERKEY = L_ORDERKEY AND L_SHIPMODE IN ('MAIL','SHIP')
AND L_COMMITDATE < L_RECEIPTDATE AND L_SHIPDATE < L_COMMITDATE AND L_RECEIPTDATE
>= '1994-01-01'
AND L_RECEIPTDATE < dateadd(mm, 1, cast('1995-09-01' as datetime))
GROUP BY L_SHIPMODE
ORDER BY L_SHIPMODE
```

QUERY 13: Customer Distribution

This query determines will determine the relationships between customers and the size of their orders;

```
SELECT C_COUNT, COUNT(*) AS CUSTDIST
FROM (SELECT C_CUSTKEY, COUNT(O_ORDERKEY)
FROM CUSTOMER left outer join ORDERS on C_CUSTKEY = O_CUSTKEY
AND O_COMMENT not like '%%special%%requests%%'
GROUP BY C_CUSTKEY) AS C_ORDERS (C_CUSTKEY, C_COUNT)
GROUP BY C_COUNT
```

```
ORDER BY CUSTDIST DESC, C_COUNT DESC
```

QUERY 14: Promotion Effect

This query will find the percentage of revenue in a year from promotional parts;

```
SELECT 100.00* SUM(CASE WHEN P_TYPE LIKE 'PROMO%%' THEN L_EXTENDEDPRICE*(1-
L_DISCOUNT) ELSE 0 END) / SUM(L_EXTENDEDPRICE*(1-L_DISCOUNT)) AS PROMO_REVENUE
FROM LINEITEM, PART
WHERE L_PARTKEY = P_PARTKEY AND L_SHIPDATE >= '1995-09-01' AND L_SHIPDATE <
dateadd(mm, 1, '1995-09-01')
```

QUERY 15: Top Supplier

This query will find the supplier that contributed the most revenue for all parts shipped during a specific time period, it includes the creation of a view to simplify the query;

```
CREATE VIEW REVENUE0 (SUPPLIER_NO, TOTAL_REVENUE) AS
SELECT L_SUPPKEY, SUM(L_EXTENDEDPRICE*(1-L_DISCOUNT)) FROM LINEITEM
WHERE L_SHIPDATE >= '1996-01-01' AND L_SHIPDATE < dateadd(mm, 3, cast('1996-01-01'
as datetime))GROUP BY L_SUPPKEY;
SELECT S_SUPPKEY, S_NAME, S_ADDRESS, S_PHONE, TOTAL_REVENUE
FROM SUPPLIER, REVENUE0
WHERE S_SUPPKEY = SUPPLIER_NO AND TOTAL_REVENUE = (SELECT MAX(TOTAL_REVENUE) FROM
REVENUE0) ORDER BY S_SUPPKEY DROP VIEW REVENUE0
```

QUERY 16: Parts/Supplier Relationship

This query will find the count of suppliers that can supply parts that meet particular customer requirements.

```
SELECT P_BRAND, P_TYPE, P_SIZE, COUNT(DISTINCT PS_SUPPKEY) AS SUPPLIER_CNT
FROM PARTSUPP, PART
WHERE P_PARTKEY = PS_PARTKEY AND P_BRAND<> 'Brand#45'
AND P_TYPE NOT LIKE 'MEDIUM POLISHED%%'AND P_SIZE IN (49, 14, 23, 45, 19, 3, 36,9)
AND PS_SUPPKEY NOT IN (SELECT S_SUPPKEY FROM SUPPLIER
WHERE S_COMMENT LIKE '%%Customer%%Complaints%%')
ORDER BY SUPPLIER_CNT DESC, P_BRAND, P_TYPE, P_SIZE
```

QUERY 17: Small-Quantity-Order Revenue

This query will find line item and part for a given brand and type and determine the average quantity of the parts ordered if the quantity is 20 percent less of the average for a seven-year period

```
SELECT SUM(L_EXTENDEDPRICE)/7.0 AS AVG_YEARLY FROM LINEITEM, PART
WHERE P_PARTKEY = L_PARTKEY AND P_BRAND = 'Brand#23' AND P_CONTAINER = 'MED BOX'
AND L_QUANTITY < (SELECT 0.2*AVG(L_QUANTITY) FROM LINEITEM WHERE L_PARTKEY =
P_PARTKEY)
```

QUERY 18: Large Volume Customer

This query will find the top 100 customers who have ever placed a large-quantity order, the quantity can be parameterized;

```
SELECT TOP 100 C_NAME, C_CUSTKEY, O_ORDERKEY, O_ORDERDATE, O_TOTALPRICE,
SUM(L_QUANTITY)
FROM CUSTOMER, ORDERS, LINEITEM
WHERE O_ORDERKEY IN (SELECT L_ORDERKEY FROM LINEITEM GROUP BY L_ORDERKEY
HAVING SUM(L_QUANTITY) > 300) AND C_CUSTKEY = O_CUSTKEY AND O_ORDERKEY =
L_ORDERKEY
GROUP BY C_NAME, C_CUSTKEY, O_ORDERKEY, O_ORDERDATE, O_TOTALPRICE
ORDER BY O_TOTALPRICE DESC, O_ORDERDATE
```

QUERY 19: Large Volume Customer

This query will find the gross discounted revenue for all orders for three different types of parts. The part type, container, quantity, ship mode, and shipping can be parameterized;

```
SELECT SUM(L_EXTENDEDPRICE* (1 - L_DISCOUNT)) AS REVENUE
FROM LINEITEM, PART
WHERE (P_PARTKEY = L_PARTKEY AND P_BRAND = 'Brand#12'
AND P_CONTAINER IN ('SM CASE', 'SM BOX', 'SM PACK', 'SM PKG') AND L_QUANTITY >= 1
AND L_QUANTITY <= 1 + 10 AND P_SIZE BETWEEN 1 AND 5
AND L_SHIPMODE IN ('AIR', 'AIR REG') AND L_SHIPINSTRUCT = 'DELIVER IN PERSON')
OR
(P_PARTKEY = L_PARTKEY AND P_BRAND ='Brand#23' AND P_CONTAINER IN ('MED BAG', 'MED
BOX', 'MED PKG', 'MED PACK') AND L_QUANTITY >=10 AND L_QUANTITY <=10 + 10 AND
P_SIZE BETWEEN 1 AND 10
AND L_SHIPMODE IN ('AIR', 'AIR REG') AND L_SHIPINSTRUCT = 'DELIVER IN PERSON')
OR
(P_PARTKEY = L_PARTKEY AND P_BRAND = 'Brand#34' AND P_CONTAINER IN ( 'LG CASE',
'LG BOX', 'LG PACK', 'LG PKG') AND L_QUANTITY >=20 AND L_QUANTITY <= 20 + 10 AND
P_SIZE BETWEEN 1 AND 15 AND L_SHIPMODE IN ('AIR', 'AIR REG') AND L_SHIPINSTRUCT =
'DELIVER IN PERSON')
```

QUERY 20: Potential Part Promotion

This query will find the suppliers that have an excess of a given part available for a specific year (the part name and date can be parameterized;

```
SELECT S_NAME, S_ADDRESS FROM SUPPLIER, NATION
WHERE S_SUPPKEY IN (SELECT PS_SUPPKEY FROM PARTSUPP
WHERE PS_PARTKEY in (SELECT P_PARTKEY FROM PART WHERE P_NAME like 'forest%%') AND
PS_AVAILQTY >  (SELECT 0.5*sum(L_QUANTITY) FROM LINEITEM WHERE L_PARTKEY =
PS_PARTKEY AND
L_SUPPKEY = PS_SUPPKEY AND L_SHIPDATE >= '1994-01-01' AND
L_SHIPDATE < dateadd (yy,1,'1994-01-01')))
AND S_NATIONKEY = N_NATIONKEY AND N_NAME = 'CANADA'
ORDER BY S_NAME
```

QUERY 21: Potential Part Promotion

This query will find the suppliers, for a given country, whose product was part of a multiple supplier order where they failed to meet the committed delivery date;

```
SELECT TOP 100 S_NAME, COUNT(*) AS NUMWAIT
FROM SUPPLIER, LINEITEM L1, ORDERS, NATION
WHERE S_SUPPKEY = L1.L_SUPPKEY AND O_ORDERKEY = L1.L_ORDERKEY AND O_ORDERSTATUS =
'F' AND L1.L_RECEIPTDATE> L1.L_COMMITDATE AND EXISTS (SELECT * FROM LINEITEM L2
WHERE L2.L_ORDERKEY = L1.L_ORDERKEY AND L2.L_SUPPKEY <> L1.L_SUPPKEY)
AND NOT EXISTS (SELECT * FROM LINEITEM L3
WHERE L3.L_ORDERKEY = L1.L_ORDERKEY AND
L3.L_SUPPKEY <> L1.L_SUPPKEY AND L3.L_RECEIPTDATE > L3.L_COMMITDATE)
AND S_NATIONKEY = N_NATIONKEY AND N_NAME = 'SAUDI ARABIA'
GROUP BY S_NAME
ORDER BY NUMWAIT DESC, S_NAME
```

QUERY 22: Global Sales Opportunity

```
SELECT CNTRYCODE, COUNT(*) AS NUMCUST, SUM(C_ACCTBAL) AS TOTACCTBAL
FROM (SELECT SUBSTRING(C_PHONE,1,2) AS CNTRYCODE, C_ACCTBAL FROM CUSTOMER WHERE
SUBSTRING(C_PHONE,1,2) IN ('13', '31', '23', '29', '30', '18', '17')
AND C_ACCTBAL > (SELECT AVG(C_ACCTBAL) FROM CUSTOMER WHERE C_ACCTBAL > 0.00 AND
SUBSTRING(C_PHONE,1,2) IN ('13', '31', '23', '29', '30', '18', '17'))
AND NOT EXISTS (SELECT * FROM ORDERS WHERE O_CUSTKEY = C_CUSTKEY)) AS CUSTSALE
GROUP BY CNTRYCODE
ORDER BY CNTRYCODE
```

Appendix G – TPC-DS Benchmark Interactive Queries

Query 3: Report the total extended sales price per item brand of a specific manufacturer for all sales in a specific month of the year;

```
select TOP 100 dt.d_year
       ,item.i_brand_id brand_id
       ,item.i_brand brand
       ,sum(ss_ext_sales_price) sum_agg
 from  date_dim dt
      ,store_sales
      ,item
 where dt.d_date_sk = store_sales.ss_sold_date_sk
   and store_sales.ss_item_sk = item.i_item_sk
   and item.i_manufact_id = 436
   and dt.d_moy=12
 group by dt.d_year
      ,item.i_brand
      ,item.i_brand_id
 order by dt.d_year
         ,sum_agg desc
         ,brand_id;
```

Query 7: Compute the average quantity, list price, discount, and sales price for promotional items sold in stores where the promotion is not offered by mail or a special event. Restrict the results to a specific gender, marital and educational status.

```
select TOP 100  i_item_id,
        avg(ss_quantity) agg1,
        avg(ss_list_price) agg2,
        avg(ss_coupon_amt) agg3,
        avg(ss_sales_price) agg4
 from store_sales, customer_demographics, date_dim, item, promotion
 where store_sales.ss_sold_date_sk = date_dim.d_date_sk and
       store_sales.ss_item_sk = item.i_item_sk and
       store_sales.ss_cdemo_sk = customer_demographics.cd_demo_sk and
       store_sales.ss_promo_sk = promotion.p_promo_sk and
       cd_gender = 'F' and
       cd_marital_status = 'W' and
       cd_education_status = 'Primary' and
       (p_channel_email = 'N' or p_channel_event = 'N') and
       d_year = 1998
 group by i_item_id
 order by i_item_id;
```

Query 12 Compute the revenue ratios across item classes: For each item in a list of given categories, during a 30 day time period, sold through the web channel compute the ratio of sales of that item to the sum of all of the sales in that item's class.

```
  select TOP 100 i_item_desc
      ,i_category
      ,i_class
      ,i_current_price
      ,i_item_id
      ,sum(ws_ext_sales_price) as itemrevenue
      ,sum(ws_ext_sales_price)*100/sum(sum(ws_ext_sales_price)) over
          (partition by i_class) as revenueratio
from
        web_sales
        ,item
        ,date_dim
where
        web_sales.ws_item_sk = item.i_item_sk
        and item.i_category in ('Jewelry', 'Sports', 'Books')
        and web_sales.ws_sold_date_sk = date_dim.d_date_sk
        and date_dim.d_date between '2001-01-12' and '2001-02-11'
group by
        i_item_id
        ,i_item_desc
        ,i_category
        ,i_class
        ,i_current_price
order by
        i_category
        ,i_class
        ,i_item_id
        ,i_item_desc
        ,revenueratio;
```

Query 15 Report the total catalog sales for customers in selected geographical regions or who made large purchases for a given year and quarter.

```
select TOP 100 ca_zip
       ,sum(cs_sales_price)
 from catalog_sales
     ,customer
     ,customer_address
     ,date_dim
 where catalog_sales.cs_bill_customer_sk = customer.c_customer_sk
        and customer.c_current_addr_sk = customer_address.ca_address_sk
        and ( substr(ca_zip,1,5) in ('85669', '86197','88274','83405','86475',
                                     '85392', '85460', '80348', '81792')
              or customer_address.ca_state in ('CA','WA','GA')
              or catalog_sales.cs_sales_price > 500)
        and catalog_sales.cs_sold_date_sk = date_dim.d_date_sk
        and date_dim.d_qoy = 2 and date_dim.d_year = 2000
 group by ca_zip
 order by ca_zip;
```

Query 18 Compute, for each county, the average quantity, list price, coupon amount, sales price, net profit, age, and number of dependents for all items purchased through catalog sales in a given year by customers who were born in a given list of six months and living in a given list of seven states and who also belong to a given gender and education demographic.

```
  select TOP 100  i_item_id,
        ca_country,
        ca_state,
        ca_county,
        avg( cast(cs_quantity as decimal(12,2))) agg1,
        avg( cast(cs_list_price as decimal(12,2))) agg2,
        avg( cast(cs_coupon_amt as decimal(12,2))) agg3,
        avg( cast(cs_sales_price as decimal(12,2))) agg4,
        avg( cast(cs_net_profit as decimal(12,2))) agg5,
        avg( cast(c_birth_year as decimal(12,2))) agg6,
        avg( cast(cd1.cd_dep_count as decimal(12,2))) agg7
 from catalog_sales, date_dim, customer_demographics cd1, item, customer,
customer_address,
      customer_demographics cd2
 where catalog_sales.cs_sold_date_sk = date_dim.d_date_sk and
       catalog_sales.cs_item_sk = item.i_item_sk and
       catalog_sales.cs_bill_cdemo_sk = cd1.cd_demo_sk and
       catalog_sales.cs_bill_customer_sk = customer.c_customer_sk and
       cd1.cd_gender = 'M' and
       cd1.cd_education_status = 'College' and
       customer.c_current_cdemo_sk = cd2.cd_demo_sk and
       customer.c_current_addr_sk = customer_address.ca_address_sk and
       c_birth_month in (9,5,12,4,1,10) and
       d_year = 2001 and
       ca_state in ('ND','WI','AL'
                   ,'NC','OK','MS','TN')
 group by i_item_id, ca_country, ca_state, ca_county with rollup
 order by ca_country,
        ca_state,
        ca_county,
        i_item_id
 limit 100;
```

Query 19 Select the top 10 revenue generating products bought by out of zip code customers for a given year, month and manager.

```
select TOP 100  i_brand_id brand_id, i_brand brand, i_manufact_id, i_manufact,
        sum(ss_ext_sales_price) ext_price
 from date_dim, store_sales, item,customer,customer_address,store
 where date_dim.d_date_sk = store_sales.ss_sold_date_sk
   and store_sales.ss_item_sk = item.i_item_sk
   and i_manager_id=7
   and d_moy=11
   and d_year=1999
   and store_sales.ss_customer_sk = customer.c_customer_sk
   and customer.c_current_addr_sk = customer_address.ca_address_sk
   and substr(ca_zip,1,5) <> substr(s_zip,1,5)
   and store_sales.ss_store_sk = store.s_store_sk
 group by i_brand
      ,i_brand_id
      ,i_manufact_id
      ,i_manufact
 order by ext_price desc
         ,i_brand
         ,i_brand_id
```

```
         ,i_manufact_id
         ,i_manufact;
```

Query 26 Computes the average quantity, list price, discount, sales price for promotional items sold through the catalog channel where the promotion was not offered by mail or in an event for given gender, marital status and educational status.

```
select TOP 100 i_item_id,
        avg(cs_quantity) agg1,
        avg(cs_list_price) agg2,
        avg(cs_coupon_amt) agg3,
        avg(cs_sales_price) agg4
 from catalog_sales, customer_demographics, date_dim, item, promotion
 where catalog_sales.cs_sold_date_sk = date_dim.d_date_sk and
       catalog_sales.cs_item_sk = item.i_item_sk and
       catalog_sales.cs_bill_cdemo_sk = customer_demographics.cd_demo_sk and
       catalog_sales.cs_promo_sk = promotion.p_promo_sk and
       cd_gender = 'F' and
       cd_marital_status = 'W' and
       cd_education_status = 'Primary' and
       (p_channel_email = 'N' or p_channel_event = 'N') and
       d_year = 1998
 group by i_item_id
 order by i_item_id;
```

Query 27 For all items sold in stores located in six states during a given year, find the average quantity, average list price, average list sales price, average coupon amount for a given gender, marital status, education and customer demographic.

```
select TOP 100 i_item_id,
        s_state,
        avg(ss_quantity) agg1,
        avg(ss_list_price) agg2,
        avg(ss_coupon_amt) agg3,
        avg(ss_sales_price) agg4
 from store_sales, customer_demographics, date_dim, store, item
 where store_sales.ss_sold_date_sk = date_dim.d_date_sk and
       store_sales.ss_item_sk = item.i_item_sk and
       store_sales.ss_store_sk = store.s_store_sk and
       store_sales.ss_cdemo_sk = customer_demographics.cd_demo_sk and
       customer_demographics.cd_gender = 'F' and
       customer_demographics.cd_marital_status = 'D' and
       customer_demographics.cd_education_status = 'Unknown' and
       date_dim.d_year = 1998 and
       store.s_state in ('KS','AL', 'MN', 'AL', 'SC', 'VT')
 group by i_item_id, s_state
 order by i_item_id
         ,s_state;
```

Query 42 For each item and a specific year and month calculate the sum of the extended sales price of store transactions.

```
 select TOP 100 dt.d_year
```

```
        ,item.i_category_id
        ,item.i_category
        ,sum(ss_ext_sales_price) as s
 from   date_dim dt
        ,store_sales
        ,item
 where dt.d_date_sk = store_sales.ss_sold_date_sk
        and store_sales.ss_item_sk = item.i_item_sk
        and item.i_manager_id = 1
        and dt.d_moy=12
        and dt.d_year=1998
 group by        dt.d_year
                 ,item.i_category_id
                 ,item.i_category
 order by       s desc,dt.d_year
                 ,item.i_category_id
                 ,item.i_category;
```

Query 43 Report the sum of all sales from Sunday to Saturday for stores in a given data range by stores.

```
select TOP 100 s_store_name, s_store_id,
        sum(case when (d_day_name='Sunday') then ss_sales_price else null end)
sun_sales,
        sum(case when (d_day_name='Monday') then ss_sales_price else null end)
mon_sales,
        sum(case when (d_day_name='Tuesday') then ss_sales_price else  null end)
tue_sales,
        sum(case when (d_day_name='Wednesday') then ss_sales_price else null end)
wed_sales,
        sum(case when (d_day_name='Thursday') then ss_sales_price else null end)
thu_sales,
        sum(case when (d_day_name='Friday') then ss_sales_price else null end)
fri_sales,
        sum(case when (d_day_name='Saturday') then ss_sales_price else null end)
sat_sales
 from date_dim, store_sales, store
 where date_dim.d_date_sk = store_sales.ss_sold_date_sk and
       store.s_store_sk = store_sales.ss_store_sk and
       s_gmt_offset = -6 and
       d_year = 1998
 group by s_store_name, s_store_id
 order by s_store_name,
s_store_id,sun_sales,mon_sales,tue_sales,wed_sales,thu_sales,fri_sales,sat_sales;
```

Query 52 Report the total of extended sales price for all items of a specific brand in a specific year and month.

```
 select TOP 100 dt.d_year
        ,item.i_brand_id brand_id
        ,item.i_brand brand
        ,sum(ss_ext_sales_price) ext_price
 from date_dim dt
     ,store_sales
     ,item
 where dt.d_date_sk = store_sales.ss_sold_date_sk
```

```
    and store_sales.ss_item_sk = item.i_item_sk
    and item.i_manager_id = 1
    and dt.d_moy=12
    and dt.d_year=1998
         group by dt.d_year
        ,item.i_brand
        ,item.i_brand_id
 order by dt.d_year
        ,ext_price desc
        ,brand_id;
```

Query 55 For a given year, month and store manager calculate the total store sales of any combination on all brands.

```
select TOP 100 i_brand_id brand_id, i_brand brand,
        sum(ss_ext_sales_price) ext_price
 from date_dim, store_sales, item
 where date_dim.d_date_sk = store_sales.ss_sold_date_sk
        and store_sales.ss_item_sk = item.i_item_sk
        and i_manager_id=36
        and d_moy=12
        and d_year=2001
 group by i_brand, i_brand_id
 order by ext_price desc, i_brand_id;
```

Query 82 Find customers who tend to spend more money (net-paid) on-line than in stores

```
select i_item_id
       ,i_item_desc
       ,i_current_price
 from item, inventory, date_dim, store_sales
 where i_current_price between 30 and 30+30
 and inv_item_sk = i_item_sk
 and d_date_sk=inv_date_sk
 and d_date between '2002-05-30' and '2002-07-30'
 and i_manufact_id in (437,129,727,663)
 and inv_quantity_on_hand between 100 and 500
 and ss_item_sk = i_item_sk
 group by i_item_id,i_item_desc,i_current_price
 order by i_item_id;
```

Query 84 List all customers living in a specified city, with an income between 2 values.

```
 select TOP 100 c_customer_id as customer_id
       ,concat(c_last_name, ', ', c_first_name) as customername
 from customer
     ,customer_address
     ,customer_demographics
     ,household_demographics
     ,income_band
     ,store_returns
 where ca_city           =  'Hopewell'
   and customer.c_current_addr_sk = customer_address.ca_address_sk
   and ib_lower_bound   >=  32287
```

```
   and ib_upper_bound   <=  32287 + 50000
   and income_band.ib_income_band_sk = household_demographics.hd_income_band_sk
   and customer_demographics.cd_demo_sk = customer.c_current_cdemo_sk
   and household_demographics.hd_demo_sk = customer.c_current_hdemo_sk
   and store_returns.sr_cdemo_sk = customer_demographics.cd_demo_sk
 order by customer_id;
```

Query 91 Display total returns of catalog sales by call center and manager in a particular month for male customers of unknown education or female customers with advanced degrees with a specified buy potential and from a particular time zone.

```
select
        cc_call_center_id Call_Center,
        cc_name Call_Center_Name,
        cc_manager Manager,
        sum(cr_net_loss) Returns_Loss
from
        call_center,
        catalog_returns,
        date_dim,
        customer,
        customer_address,
        customer_demographics,
        household_demographics
where
        catalog_returns.cr_call_center_sk       = call_center.cc_call_center_sk
and     catalog_returns.cr_returned_date_sk     = date_dim.d_date_sk
and     catalog_returns.cr_returning_customer_sk= customer.c_customer_sk
and     customer_demographics.cd_demo_sk              =
customer.c_current_cdemo_sk
and     household_demographics.hd_demo_sk              =
customer.c_current_hdemo_sk
and     customer_address.ca_address_sk           = customer.c_current_addr_sk
and     d_year                  = 1999
and     d_moy                   = 11
and     ( (cd_marital_status       = 'M' and cd_education_status     = 'Unknown')
        or(cd_marital_status       = 'W' and cd_education_status     = 'Advanced
Degree'))
and     hd_buy_potential like '0-500%'
and     ca_gmt_offset           = -7
group by
cc_call_center_id,cc_name,cc_manager,cd_marital_status,cd_education_status
order by Returns_Loss desc;
```

Query 96 Compute a count of sales from a named store to customers with a given number of dependents made in a specified half hour period of the day.

```
select TOP 100 count(*) as c
from store_sales
    ,household_demographics
    ,time_dim, store
where store_sales.ss_sold_time_sk = time_dim.t_time_sk
    and store_sales.ss_hdemo_sk = household_demographics.hd_demo_sk
    and store_sales.ss_store_sk = store.s_store_sk
    and time_dim.t_hour = 8
```

```
    and time_dim.t_minute >= 30
    and household_demographics.hd_dep_count = 5
    and store.s_store_name = 'ese'
order by c;
```

Appendix H – TPC-DS Benchmark Reporting Queries

Query 17: Analyze, for each state, all items that were sold in stores in a particular quarter and returned in the next three quarters and then re-purchased by the customer through the catalog channel in the three following quarters.

```
select TOP 100 i_item_id
       ,i_item_desc
       ,s_state
       ,count(ss_quantity) as store_sales_quantitycount
       ,avg(ss_quantity) as store_sales_quantityave
       ,stddev_samp(ss_quantity) as store_sales_quantitystdev
       ,stddev_samp(ss_quantity)/avg(ss_quantity) as store_sales_quantitycov
       ,count(sr_return_quantity) as_store_returns_quantitycount
       ,avg(sr_return_quantity) as_store_returns_quantityave
       ,stddev_samp(sr_return_quantity) as_store_returns_quantitystdev
       ,stddev_samp(sr_return_quantity)/avg(sr_return_quantity) as
store_returns_quantitycov
       ,count(cs_quantity) as catalog_sales_quantitycount ,avg(cs_quantity) as
catalog_sales_quantityave
       ,stddev_samp(cs_quantity)/avg(cs_quantity) as catalog_sales_quantitystdev
       ,stddev_samp(cs_quantity)/avg(cs_quantity) as catalog_sales_quantitycov
 from store_sales
     ,store_returns
     ,catalog_sales
     ,date_dim d1
     ,date_dim d2
     ,date_dim d3
     ,store
     ,item
 where d1.d_quarter_name = '2000Q1'
   and d1.d_date_sk = store_sales.ss_sold_date_sk
   and item.i_item_sk = store_sales.ss_item_sk
   and store.s_store_sk = store_sales.ss_store_sk
   and store_sales.ss_customer_sk = store_returns.sr_customer_sk
   and store_sales.ss_item_sk = store_returns.sr_item_sk
   and store_sales.ss_ticket_number = store_returns.sr_ticket_number
   and store_returns.sr_returned_date_sk = d2.d_date_sk
   and d2.d_quarter_name in ('2000Q1','2000Q2','2000Q3')
   and store_returns.sr_customer_sk = catalog_sales.cs_bill_customer_sk
   and store_returns.sr_item_sk = catalog_sales.cs_item_sk
   and catalog_sales.cs_sold_date_sk = d3.d_date_sk
   and d3.d_quarter_name in ('2000Q1','2000Q2','2000Q3')
 group by i_item_id
         ,i_item_desc
         ,s_state
 order by i_item_id
         ,i_item_desc
         ,s_state;
```

Query 21: For all items whose price was changed on a given date, compute the percentage change in inventory between the 30-day period BEFORE the price change and the 30-day period AFTER the change. Group this information by warehouse.

```
select TOP 100 *
 from(select w_warehouse_name
            ,i_item_id
            ,sum(case when (cast(d_date as date) < cast ('1998-04-08' as date))
                        then inv_quantity_on_hand
                      else 0 end) as inv_before
            ,sum(case when (cast(d_date as date) >= cast ('1998-04-08' as date))
                      then inv_quantity_on_hand
                      else 0 end) as inv_after
   from inventory
       ,warehouse
       ,item
       ,date_dim
   where i_current_price between 0.99 and 1.49
     and item.i_item_sk          = inventory.inv_item_sk
     and inventory.inv_warehouse_sk   = warehouse.w_warehouse_sk
     and inventory.inv_date_sk    = date_dim.d_date_sk
     and d_date between '1998-03-09' and '1998-05-07'
   group by w_warehouse_name, i_item_id) x
 where (case when inv_before > 0
             then inv_after / inv_before
             else null
             end) between 2.0/3.0 and 3.0/2.0
 order by w_warehouse_name
         ,i_item_id;
```

Query 32 Compute the total discounted amount for a particular manufacturer in a particular 90 day period for catalog sales whose discounts exceeded the average discount by at least 30%

```
  SELECT sum(cs1.cs_ext_discount_amt) as excess_discount_amount
FROM (SELECT cs.cs_item_sk as cs_item_sk,
                            cs.cs_ext_discount_amt as cs_ext_discount_amt
             FROM catalog_sales cs
             JOIN date_dim d ON (d.d_date_sk = cs.cs_sold_date_sk)
             WHERE d.d_date between '2000-01-27' and '2000-04-27') cs1
JOIN item i ON (i.i_item_sk = cs1.cs_item_sk)
JOIN (SELECT cs2.cs_item_sk as cs_item_sk,
                          1.3 * avg(cs_ext_discount_amt) as
avg_cs_ext_discount_amt
           FROM (SELECT cs.cs_item_sk as cs_item_sk,
                                        cs.cs_ext_discount_amt as
cs_ext_discount_amt
                        FROM catalog_sales cs
                        JOIN date_dim d ON (d.d_date_sk = cs.cs_sold_date_sk)
                        WHERE d.d_date between '2000-01-27' and '2000-04-27') cs2
                        GROUP BY cs2.cs_item_sk) tmp1
ON (i.i_item_sk = tmp1.cs_item_sk)
WHERE i.i_manufact_id = 436 and
               cs1.cs_ext_discount_amt > tmp1.avg_cs_ext_discount_amt;
```

Query 40 Compute the impact of an item price change on the sales by computing the total sales for items in a 30 day period before and after the price change. Group the items by location of warehouse where they were delivered from.

```
select TOP 100
   w_state
  ,i_item_id
  ,sum(case when (cast(d_date as date) < cast ('1998-04-08' as date))
                then cs_sales_price - coalesce(cr_refunded_cash,0) else 0 end) as
sales_before
  ,sum(case when (cast(d_date as date) >= cast ('1998-04-08' as date))
                then cs_sales_price - coalesce(cr_refunded_cash,0) else 0 end) as
sales_after
 from
   catalog_sales left outer join catalog_returns on
       (catalog_sales.cs_order_number = catalog_returns.cr_order_number
        and catalog_sales.cs_item_sk = catalog_returns.cr_item_sk)
  ,warehouse
  ,item
  ,date_dim
 where
     i_current_price between 0.99 and 1.49
 and item.i_item_sk          = catalog_sales.cs_item_sk
 and catalog_sales.cs_warehouse_sk    = warehouse.w_warehouse_sk
 and catalog_sales.cs_sold_date_sk    = date_dim.d_date_sk
 and date_dim.d_date between '1998-03-09' and '1998-05-08'
 group by
    w_state,i_item_id
 order by w_state,i_item_id;
```

Query 46 Compute the per-customer coupon amount and net profit of all "out of town" customers buying from stores located in 5 cities on weekends in three consecutive years. The customers need to fit the profile of having a specific dependent count and vehicle count. For all these customers print the city they lived in at the time of purchase, the city in which the store is located, the coupon amount and net profit

```
select TOP 100 c_last_name
       ,c_first_name
       ,ca_city
       ,bought_city
       ,ss_ticket_number
       ,amt,profit
 from
   (select ss_ticket_number
          ,ss_customer_sk
          ,ca_city bought_city
          ,sum(ss_coupon_amt) amt
          ,sum(ss_net_profit) profit
    from store_sales,date_dim,store,household_demographics,customer_address
    where store_sales.ss_sold_date_sk = date_dim.d_date_sk
    and store_sales.ss_store_sk = store.s_store_sk
    and store_sales.ss_hdemo_sk = household_demographics.hd_demo_sk
    and store_sales.ss_addr_sk = customer_address.ca_address_sk
    and (household_demographics.hd_dep_count = 4 or
         household_demographics.hd_vehicle_count= 2)
```

```
    and date_dim.d_dow in (6,0)
    and date_dim.d_year in (1998,1998+1,1998+2)
    and store.s_city in ('Rosedale','Bethlehem','Clinton','Clifton','Springfield')
    group by ss_ticket_number,ss_customer_sk,ss_addr_sk,ca_city)
dn,customer,customer_address current_addr
    where dn.ss_customer_sk = customer.c_customer_sk
      and customer.c_current_addr_sk = current_addr.ca_address_sk
      and current_addr.ca_city <> bought_city
  order by c_last_name
          ,c_first_name
          ,ca_city
          ,bought_city
          ,ss_ticket_number;
```

Query 58 Retrieve the items generating the highest revenue and which had a revenue that was approximately equivalent across all of store, catalog and web within the week ending a given date.

```
select TOP 100 ss_items.item_id
       ,ss_item_rev
       ,ss_item_rev/(ss_item_rev+cs_item_rev+ws_item_rev)/3 * 100 ss_dev
       ,cs_item_rev
       ,cs_item_rev/(ss_item_rev+cs_item_rev+ws_item_rev)/3 * 100 cs_dev
       ,ws_item_rev
       ,ws_item_rev/(ss_item_rev+cs_item_rev+ws_item_rev)/3 * 100 ws_dev
       ,(ss_item_rev+cs_item_rev+ws_item_rev)/3 average
FROM
( select i_item_id item_id ,sum(ss_ext_sales_price) as ss_item_rev
 from store_sales
     JOIN item ON store_sales.ss_item_sk = item.i_item_sk
     JOIN date_dim ON store_sales.ss_sold_date_sk = date_dim.d_date_sk
     JOIN (select d1.d_date
                 from date_dim d1 JOIN date_dim d2 ON d1.d_week_seq =
d2.d_week_seq
                 where d2.d_date = '1998-08-04') sub ON date_dim.d_date =
sub.d_date
 group by i_item_id ) ss_items
JOIN
( select i_item_id item_id ,sum(cs_ext_sales_price) as cs_item_rev
 from catalog_sales
     JOIN item ON catalog_sales.cs_item_sk = item.i_item_sk
     JOIN date_dim ON catalog_sales.cs_sold_date_sk = date_dim.d_date_sk
     JOIN (select d1.d_date
                 from date_dim d1 JOIN date_dim d2 ON d1.d_week_seq =
d2.d_week_seq
                 where d2.d_date = '1998-08-04') sub ON date_dim.d_date =
sub.d_date
 group by i_item_id ) cs_items
ON ss_items.item_id=cs_items.item_id
JOIN
( select i_item_id item_id ,sum(ws_ext_sales_price) as ws_item_rev
 from web_sales
     JOIN item ON web_sales.ws_item_sk = item.i_item_sk
     JOIN date_dim ON web_sales.ws_sold_date_sk = date_dim.d_date_sk
     JOIN (select d1.d_date
                 from date_dim d1 JOIN date_dim d2 ON d1.d_week_seq =
d2.d_week_seq
```

```
                  where d2.d_date = '1998-08-04') sub ON date_dim.d_date =
sub.d_date
 group by i_item_id ) ws_items
ON ss_items.item_id=ws_items.item_id
 where
       ss_item_rev between 0.9 * cs_item_rev and 1.1 * cs_item_rev
   and ss_item_rev between 0.9 * ws_item_rev and 1.1 * ws_item_rev
   and cs_item_rev between 0.9 * ss_item_rev and 1.1 * ss_item_rev
   and cs_item_rev between 0.9 * ws_item_rev and 1.1 * ws_item_rev
   and ws_item_rev between 0.9 * ss_item_rev and 1.1 * ss_item_rev
   and ws_item_rev between 0.9 * cs_item_rev and 1.1 * cs_item_rev
 order by item_id ,ss_item_rev;
```

Query 68 Compute the per customer extended sales price, extended list price and extended tax for "out of town" shoppers buying from stores located in two cities in the first two days of each month of three consecutive years. Only consider customers with specific dependent and vehicle counts.

```
select TOP 100 c_last_name
       ,c_first_name
       ,ca_city
       ,bought_city
       ,ss_ticket_number
       ,extended_price
       ,extended_tax
       ,list_price
 from (select ss_ticket_number
             ,ss_customer_sk
             ,ca_city bought_city
             ,sum(ss_ext_sales_price) extended_price
             ,sum(ss_ext_list_price) list_price
             ,sum(ss_ext_tax) extended_tax
       from store_sales
           ,date_dim
           ,store
           ,household_demographics
           ,customer_address
       where store_sales.ss_sold_date_sk = date_dim.d_date_sk
         and store_sales.ss_store_sk = store.s_store_sk
        and store_sales.ss_hdemo_sk = household_demographics.hd_demo_sk
        and store_sales.ss_addr_sk = customer_address.ca_address_sk
        and date_dim.d_dom between 1 and 2
        and (household_demographics.hd_dep_count = 4 or
             household_demographics.hd_vehicle_count= 2)
        and date_dim.d_year in (1998,1998+1,1998+2)
        and store.s_city in ('Rosedale','Bethlehem')
       group by ss_ticket_number
               ,ss_customer_sk
               ,ss_addr_sk,ca_city) dn
      ,customer
      ,customer_address current_addr
 where dn.ss_customer_sk = customer.c_customer_sk
   and customer.c_current_addr_sk = current_addr.ca_address_sk
   and current_addr.ca_city <> bought_city
 order by c_last_name
         ,ss_ticket_number;
```

Query 76 Computes the average quantity, list price, discount, sales price for promotional items sold through the web channel where the promotion is not offered by mail or in an event for given gender, marital status and educational status.

```
select TOP 100 channel, col_name, d_year, d_qoy, i_category, COUNT(*) sales_cnt,
SUM(ext_sales_price) sales_amt FROM (
        SELECT 'store' as channel, 'ss_addr_sk' col_name, d_year, d_qoy,
i_category, ss_ext_sales_price ext_sales_price
         FROM store_sales, item, date_dim
         WHERE ss_addr_sk IS NULL
           AND store_sales.ss_sold_date_sk=date_dim.d_date_sk
           AND store_sales.ss_item_sk=item.i_item_sk
        UNION ALL
        SELECT 'web' as channel, 'ws_web_page_sk' col_name, d_year, d_qoy,
i_category, ws_ext_sales_price ext_sales_price
         FROM web_sales, item, date_dim
         WHERE ws_web_page_sk IS NULL
           AND web_sales.ws_sold_date_sk=date_dim.d_date_sk
           AND web_sales.ws_item_sk=item.i_item_sk
        UNION ALL
        SELECT 'catalog' as channel, 'cs_warehouse_sk' col_name, d_year, d_qoy,
i_category, cs_ext_sales_price ext_sales_price
         FROM catalog_sales, item, date_dim
         WHERE cs_warehouse_sk IS NULL
           AND catalog_sales.cs_sold_date_sk=date_dim.d_date_sk
           AND catalog_sales.cs_item_sk=item.i_item_sk) foo
GROUP BY channel, col_name, d_year, d_qoy, i_category
ORDER BY channel, col_name, d_year, d_qoy, i_category;
```

Query 79 Compute the per customer coupon amount and net profit of Monday shoppers. Only purchases of three consecutive years made on Mondays in large stores by customers with a certain dependent count and with a large vehicle count are considered.

```
select TOP 100
  c_last_name,c_first_name,substr(s_city,1,30) sub,ss_ticket_number,amt,profit
  from
   (select ss_ticket_number
          ,ss_customer_sk
          ,store.s_city
          ,sum(ss_coupon_amt) amt
          ,sum(ss_net_profit) profit
    from store_sales,date_dim,store,household_demographics
    where store_sales.ss_sold_date_sk = date_dim.d_date_sk
    and store_sales.ss_store_sk = store.s_store_sk
    and store_sales.ss_hdemo_sk = household_demographics.hd_demo_sk
    and (household_demographics.hd_dep_count = 8 or
household_demographics.hd_vehicle_count > 0)
    and date_dim.d_dow = 1
    and date_dim.d_year in (1998,1998+1,1998+2)
    and store.s_number_employees between 200 and 295
    group by ss_ticket_number,ss_customer_sk,ss_addr_sk,store.s_city) ms,customer
    where ms.ss_customer_sk = customer.c_customer_sk
 order by c_last_name,c_first_name,sub, profit;
```

Query 88 How many items do we sell between pacific times of a day in certain stores to customers with one dependent count and 2 or less vehicles registered or 2 dependents with 4 or fewer vehicles registered or 3 dependents and five or less vehicles registered. In one row break the counts into sells from 8:30 to 9, 9 to 9:30, 9:30 to 10 ... 12 to 12:30.

```
select  *
from
 (select count(*) h8_30_to_9
 from store_sales, household_demographics , time_dim, store
 where store_sales.ss_sold_time_sk = time_dim.t_time_sk
     and store_sales.ss_hdemo_sk = household_demographics.hd_demo_sk
     and store_sales.ss_store_sk = store.s_store_sk
     and time_dim.t_hour = 8
     and time_dim.t_minute >= 30
     and ((household_demographics.hd_dep_count = 3 and
household_demographics.hd_vehicle_count<=3+2) or
          (household_demographics.hd_dep_count = 0 and
household_demographics.hd_vehicle_count<=0+2) or
          (household_demographics.hd_dep_count = 1 and
household_demographics.hd_vehicle_count<=1+2))
     and store.s_store_name = 'ese') s1,
 (select count(*) h9_to_9_30
 from store_sales, household_demographics , time_dim, store
 where store_sales.ss_sold_time_sk = time_dim.t_time_sk
     and store_sales.ss_hdemo_sk = household_demographics.hd_demo_sk
     and store_sales.ss_store_sk = store.s_store_sk
     and time_dim.t_hour = 9
     and time_dim.t_minute < 30
     and ((household_demographics.hd_dep_count = 3 and
household_demographics.hd_vehicle_count<=3+2) or
          (household_demographics.hd_dep_count = 0 and
household_demographics.hd_vehicle_count<=0+2) or
          (household_demographics.hd_dep_count = 1 and
household_demographics.hd_vehicle_count<=1+2))
     and store.s_store_name = 'ese') s2,
 (select count(*) h9_30_to_10
 from store_sales, household_demographics , time_dim, store
 where store_sales.ss_sold_time_sk = time_dim.t_time_sk
     and store_sales.ss_hdemo_sk = household_demographics.hd_demo_sk
     and store_sales.ss_store_sk = store.s_store_sk
     and time_dim.t_hour = 9
     and time_dim.t_minute >= 30
     and ((household_demographics.hd_dep_count = 3 and
household_demographics.hd_vehicle_count<=3+2) or
          (household_demographics.hd_dep_count = 0 and
household_demographics.hd_vehicle_count<=0+2) or
          (household_demographics.hd_dep_count = 1 and
household_demographics.hd_vehicle_count<=1+2))
     and store.s_store_name = 'ese') s3,
 (select count(*) h10_to_10_30
 from store_sales, household_demographics , time_dim, store
 where store_sales.ss_sold_time_sk = time_dim.t_time_sk
     and store_sales.ss_hdemo_sk = household_demographics.hd_demo_sk
     and store_sales.ss_store_sk = store.s_store_sk
     and time_dim.t_hour = 10
     and time_dim.t_minute < 30
     and ((household_demographics.hd_dep_count = 3 and
household_demographics.hd_vehicle_count<=3+2) or
```

```
          (household_demographics.hd_dep_count = 0 and
household_demographics.hd_vehicle_count<=0+2) or
          (household_demographics.hd_dep_count = 1 and
household_demographics.hd_vehicle_count<=1+2))
     and store.s_store_name = 'ese') s4,
 (select count(*) h10_30_to_11
 from store_sales, household_demographics , time_dim, store
 where store_sales.ss_sold_time_sk = time_dim.t_time_sk
     and store_sales.ss_hdemo_sk = household_demographics.hd_demo_sk
     and store_sales.ss_store_sk = store.s_store_sk
     and time_dim.t_hour = 10
     and time_dim.t_minute >= 30
     and ((household_demographics.hd_dep_count = 3 and
household_demographics.hd_vehicle_count<=3+2) or
          (household_demographics.hd_dep_count = 0 and
household_demographics.hd_vehicle_count<=0+2) or
          (household_demographics.hd_dep_count = 1 and
household_demographics.hd_vehicle_count<=1+2))
     and store.s_store_name = 'ese') s5,
 (select count(*) h11_to_11_30
 from store_sales, household_demographics , time_dim, store
 where store_sales.ss_sold_time_sk = time_dim.t_time_sk
     and store_sales.ss_hdemo_sk = household_demographics.hd_demo_sk
     and store_sales.ss_store_sk = store.s_store_sk
     and time_dim.t_hour = 11
     and time_dim.t_minute < 30
     and ((household_demographics.hd_dep_count = 3 and
household_demographics.hd_vehicle_count<=3+2) or
          (household_demographics.hd_dep_count = 0 and
household_demographics.hd_vehicle_count<=0+2) or
          (household_demographics.hd_dep_count = 1 and
household_demographics.hd_vehicle_count<=1+2))
     and store.s_store_name = 'ese') s6,
 (select count(*) h11_30_to_12
 from store_sales, household_demographics , time_dim, store
 where store_sales.ss_sold_time_sk = time_dim.t_time_sk
     and store_sales.ss_hdemo_sk = household_demographics.hd_demo_sk
     and store_sales.ss_store_sk = store.s_store_sk
     and time_dim.t_hour = 11
     and time_dim.t_minute >= 30
     and ((household_demographics.hd_dep_count = 3 and
household_demographics.hd_vehicle_count<=3+2) or
          (household_demographics.hd_dep_count = 0 and
household_demographics.hd_vehicle_count<=0+2) or
          (household_demographics.hd_dep_count = 1 and
household_demographics.hd_vehicle_count<=1+2))
     and store.s_store_name = 'ese') s7,
 (select count(*) h12_to_12_30
 from store_sales, household_demographics , time_dim, store
 where store_sales.ss_sold_time_sk = time_dim.t_time_sk
     and store_sales.ss_hdemo_sk = household_demographics.hd_demo_sk
     and store_sales.ss_store_sk = store.s_store_sk
     and time_dim.t_hour = 12
     and time_dim.t_minute < 30
     and ((household_demographics.hd_dep_count = 3 and
household_demographics.hd_vehicle_count<=3+2) or
          (household_demographics.hd_dep_count = 0 and
household_demographics.hd_vehicle_count<=0+2) or
```

```
          (household_demographics.hd_dep_count = 1 and
household_demographics.hd_vehicle_count<=1+2))
     and store.s_store_name = 'ese') s8;
```

Query 90 What is the ratio between the number of items sold over the internet in the morning (8 to 9am) to the number of items sold in the evening (7 to 8pm) of customers with a specified number of dependents. Consider only websites with a high amount of content.

```
select cast(amc as decimal(15,4))/cast(pmc as decimal(15,4)) am_pm_ratio
 from ( select count(*) amc
       from web_sales, household_demographics , time_dim, web_page
       where ws_sold_time_sk = time_dim.t_time_sk
         and ws_ship_hdemo_sk = household_demographics.hd_demo_sk
         and ws_web_page_sk = web_page.wp_web_page_sk
         and time_dim.t_hour between 6 and 6+1
         and household_demographics.hd_dep_count = 8
         and web_page.wp_char_count between 5000 and 5200) at,
      ( select count(*) pmc
       from web_sales, household_demographics , time_dim, web_page
       where ws_sold_time_sk = time_dim.t_time_sk
         and ws_ship_hdemo_sk = household_demographics.hd_demo_sk
         and ws_web_page_sk = web_page.wp_web_page_sk
         and time_dim.t_hour between 14 and 14+1
         and household_demographics.hd_dep_count = 8
         and web_page.wp_char_count between 5000 and 5200) pt
 order by am_pm_ratio
 LIMIT 100;
```

Query 92 Compute the total discount on web sales of items from a given manufacturer over a particular 90 day period for sales whose discount exceeded 30% over the average discount of items from that manufacturer in that period of time.

```
SELECT sum(case when ssci.customer_sk is not null and csci.customer_sk is null
then 1
                                  else 0 end) as store_only,
               sum(case when ssci.customer_sk is null and csci.customer_sk is not
null then 1
                                 else 0 end) as catalog_only,
               sum(case when ssci.customer_sk is not null and csci.customer_sk is
not null then 1
                                  else 0 end) as store_and_catalog
FROM (SELECT ss.ss_customer_sk as customer_sk,
                              ss.ss_item_sk as item_sk
             FROM store_sales ss
             JOIN date_dim d1 ON (ss.ss_sold_date_sk = d1.d_date_sk)
             WHERE d1.d_month_seq >= 1206 and
                            d1.d_month_seq <= 1217
             GROUP BY ss.ss_customer_sk, ss.ss_item_sk) ssci
FULL OUTER JOIN (SELECT cs.cs_bill_customer_sk as customer_sk,
                                                  cs.cs_item_sk as item_sk
                                    FROM catalog_sales cs
                                    JOIN date_dim d2 ON (cs.cs_sold_date_sk =
d2.d_date_sk)
                                    WHERE d2.d_month_seq >= 1206 and
                                                  d2.d_month_seq <= 1217
```

```
                                    GROUP BY cs.cs_bill_customer_sk, cs.cs_item_sk)
csci
ON (ssci.customer_sk=csci.customer_sk and
        ssci.item_sk = csci.item_sk);
```

Query 93 For a given merchandise return reason, report on customers' total cost of purchases minus the cost of returned items.

```
select TOP 100 ss_customer_sk
            ,sum(act_sales) sumsales
      from (select ss_item_sk
                  ,ss_ticket_number
                  ,ss_customer_sk
                  ,case when sr_return_quantity is not null then (ss_quantity-
sr_return_quantity)*ss_sales_price
                                                              else
(ss_quantity*ss_sales_price) end act_sales
            from store_sales left outer join store_returns on
(store_returns.sr_item_sk = store_sales.ss_item_sk
                                                                and
store_returns.sr_ticket_number = store_sales.ss_ticket_number)
                 ,reason
            where store_returns.sr_reason_sk = reason.r_reason_sk
              and r_reason_desc = 'Did not like the warranty') t
      group by ss_customer_sk
      order by sumsales, ss_customer_sk;
```

Query 95 Produce a count of web sales and total shipping cost and net profit in a given 60 day period to customers in a given state from a named web site for returned orders shipped from more than one warehouse.

```
SELECT count(distinct ws1.ws_order_number) as order_count,
               sum(ws1.ws_ext_ship_cost) as total_shipping_cost,
               sum(ws1.ws_net_profit) as total_net_profit
FROM web_sales ws1
JOIN customer_address ca ON (ws1.ws_ship_addr_sk = ca.ca_address_sk)
JOIN web_site s ON (ws1.ws_web_site_sk = s.web_site_sk)
JOIN date_dim d ON (ws1.ws_ship_date_sk = d.d_date_sk)
LEFT SEMI JOIN (SELECT ws2.ws_order_number as ws_order_number
                               FROM web_sales ws2 JOIN web_sales ws3
                               ON (ws2.ws_order_number = ws3.ws_order_number)
                               WHERE ws2.ws_warehouse_sk <> ws3.ws_warehouse_sk
                          ) ws_wh1
ON (ws1.ws_order_number = ws_wh1.ws_order_number)
LEFT SEMI JOIN (SELECT wr_order_number
                               FROM web_returns wr
                               JOIN (SELECT ws4.ws_order_number as ws_order_number
                                         FROM web_sales ws4 JOIN web_sales ws5
                                         ON (ws4.ws_order_number =
ws5.ws_order_number)
                                        WHERE ws4.ws_warehouse_sk <>
ws5.ws_warehouse_sk
                                  ) ws_wh2
                               ON (wr.wr_order_number = ws_wh2.ws_order_number))
tmp1
ON (ws1.ws_order_number = tmp1.wr_order_number)
WHERE d.d_date between '2002-05-01' and '2002-06-30' and
               ca.ca_state = 'GA' and
```

```
                s.web_company_name = 'pri';
```

Query 97 Generate counts of promotional sales and total sales, and their ratio from the web channel for a particular item category and month to customers in a given time zone.

```
select TOP 200 sum(case when ssci.customer_sk is not null and csci.customer_sk is
null then 1 else 0 end) store_only
      ,sum(case when ssci.customer_sk is null and csci.customer_sk is not null
then 1 else 0 end) catalog_only
      ,sum(case when ssci.customer_sk is not null and csci.customer_sk is not null
then 1 else 0 end) store_and_catalog
from
( select ss_customer_sk customer_sk
      ,ss_item_sk item_sk
from store_sales
JOIN date_dim ON store_sales.ss_sold_date_sk = date_dim.d_date_sk
where
  d_month_seq between 1193 and 1193 + 11
group by ss_customer_sk ,ss_item_sk) ssci
full outer join
( select cs_bill_customer_sk customer_sk
      ,cs_item_sk item_sk
from catalog_sales
JOIN date_dim ON catalog_sales.cs_sold_date_sk = date_dim.d_date_sk
where
  d_month_seq between 1193 and 1193 + 11
group by cs_bill_customer_sk ,cs_item_sk) csci
on (ssci.customer_sk=csci.customer_sk and ssci.item_sk = csci.item_sk);
```