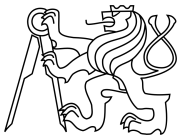
CZECH TECHNICAL UNIVERSITY IN PRAGUE
FACULTY OF INFORMATION TECHNOLOGY



ASSIGNMENT OF BACHELOR'S THESIS

Title:	SQL Engines for Big Data Analytics; Comparison and its Usage in Business Field
Student:	Elena Agapova
Supervisor:	Ing. Michal Valenta, Ph.D.
Study Programme:	Informatics
Study Branch:	Information Systems and Management
Department:	Department of Software Engineering
Validity:	Until the end of summer semester 2018/19

Instructions

The aim of this work is to investigate a rapidly growing market of SQL engines used for big data analytics and to define which of them should be used under different businesses requirements.

1. Describe how has the concept of big data had influenced the data analytics field, learn its history and the last trends. Describe a big data architecture, its components and how does the big data platform work for data analytics purposes.

2. Study existing approaches of using SQL in big data. Define a list of big data SQL engines you will focus on in the rest of your work. Justify your choice using criteria defined together with your supervisor.

3. Compare SQL engines defined in step 2 according to business and technical criteria and specifications (the most frequent use cases, existing business model, implementation costs, speed, security etc.)

4. Make a final report collecting all pros and cons and a brief conclusion considering the best usage of every analyzed SQL engine.

References

Will be provided by the supervisor.

Ing. Michal Valenta, Ph.D.
Head of Department

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
Dean

Prague October 12, 2017

Czech Technical University in Prague
Faculty of Information Technology
Department of Software Engineering

Bachelor's thesis

SQL Engines for Big Data Analytics; Comparison and its Usage in Business Field

Elena Agapova

Supervisor: Ing. Michal Valenta, Ph.D.

10th May 2018

Acknowledgements

I would like to thank my supervisor, Ing. Michal Valenta, Ph.D., for his guidance and assistance while working on this thesis. I would also like to express my gratitude to my family and friends for their encouragement and support during my studies.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as school work under the provisions of Article 60(1) of the Act.

In Prague on 10th May 2018 .....

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Agapova, Elena. *SQL Engines for Big Data Analytics; Comparison and its Usage in Business Field*. Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2018.

Abstrakt

Díky rostoucím objemům dat generovaným v dnešní době, jejich vývoji a změnám, koncept velkých dat (big data) se stal objektem velkého zájmu. Tento koncept popisuje sadu technik a metod pro efektivní práci s velkými soubory dat.

V moderním světě informace je jedním z nejdůležitějších prostředků, který může přispívat k optimalizaci podnikových procesů a rozhodování. Nicméně, pro získání přehledu a využití potenciálu dat je nutné je zpracovat a zjistit jaká informace se skrývá uvnitř, k čemuž slouží analýza dat. Platformy pro analýzu dat umožňují manipulovat s datami pro zjištění jejich přesného obsahu. Podobné systémy jsou velmi komplexní a obsahují více jednotlivých komponentů a proto jejich návrh vyžaduje rozsáhlý průzkum dostupných možností.

Tato bakalářská práce má za cíl popsat jak analýza dat funguje pro velká data a jejich společné využití v praxi. Zaměřuje se na popis existujících technologií a přístupů k realizaci infrastruktur pro datovou analýzu. Práce se také zabývá definicí faktorů které by měly být vzaté v úvahu při výběru nejvhodnějšího softwarového řešení pro konkrétní případy užití.

Výzkum je proveden pomocí studia architektonických základů systémů pro zpracování velkých dat a průzkumů trhu softwaru pro analýzu dat. Výsledkem prací je zpráva porovnávající vybrané technologie a popisující jednotlivé kritéria, které byly vzaté v úvahu při porovnání. Tato zpráva může být použita jako návod k výběru vhodného softwarového řešení pro implementaci analytického systému v různých podnicích.

Klíčová slova Big data, analýza velkých dat, databázové stroje, platformy pro analýzu dat, porovnání technologií

Abstract

With a growing amount of data generated, their changing and evolving, the concept of big data has become incredibly popular in last years. It provides a set of new approaches and techniques allowing to work efficiently with huge volumes of records.

Nowadays, information is one of the most important resources; it can help with decision making and business processes optimization. However, to get actual insights and unlock a potential of data, it is necessary to process them and discover the information hidden inside it which is a goal of data analytics. Data analytic platforms allow to manipulate with raw data in order to find out what exactly they contain. These systems are complex and includes multiple components therefore their designing requires comprehensive analysis of available options.

This thesis aims to describe how data analytics works for big data and how they are used in business. It gives an overview of existing technologies and approaches to building data analytics infrastructures. It also defines points that should be taken into consideration while choosing the most suitable software solution for a particular use case.

The research is done by studying architectural principles of big data systems and investigating the market of data analytics software. The result of this work is a composite report including comparison of several technologies and a list of criteria considered. The final report can be used as a guideline for choosing the most suitable technology for implementing an analytical platform in a broad variety of organizations.

Keywords Big data, big data analytics, SQL engine, data analytics platform, technologies comparison

Contents

List of Figures

Introduction

The amount of data existing in the world is growing rapidly nowadays with an increasing number of machines and systems producing them. Data are becoming more diverse, uneven, mixed; they are changing and evolving together with overall technologies landscape.

However, the existence of data by itself does not make any real impact; to take an advantage of having them, it is necessary to process it to discover what is hidden inside. The true value of data is in information they contain as it can give insights into business processes and customer behavior, help to predict changes and make prognoses about the future. Understanding the potential of data and leveraging it helps to optimize organizational activities and operations which in turn brings competitive advantage and make business more successful and profitable. It also unlocks an opportunity to enable a data-driven approach when decisions are taken based on facts and verified information derived from them.

Effective leveraging of data analytics techniques requires having an infrastructure for consuming, storing and processing data with following submitting obtained results to end users. These systems usually consist of data storages, SQL engines and reporting tools integrated together to create a seamless pipeline taking raw data as an input and returning results of analytic queries executed on it. Designing such systems brings a lot of challenges as it is necessary not only to pick up suitable components but also ensure that they can work together.

This work is focused on SQL engines particularly as they are de-facto the key element for analytical systems since it is the component which actually processes the data and helps to discover what they contain. SQL engines execute queries sent by end users for getting specific insights answering their questions or providing them with findings which can help to do so.

Nowadays market offers dozens of software solutions for building data analytics infrastructure for any workloads and a big variety of options available does not make the choice easier; contrarily, it may be confusing or even over-

whelming. Moreover, big data ideas are becoming more and more popular and today leveraging it is already considered a necessity rather than a nice-to-have option.

The goal of this paper is to provide an overview of how do big data analytics workflows function in general and to summarize main aspects important for choosing the right engine capable of efficiently solving given tasks. The research is done by exploring the big data technology market together with studying architectural principles of data analytics systems. The resulting report includes the table comparing the chosen engines, detailed explanations of criteria considered in examination and example of use cases for each of them. This report can be used as a guideline for choosing a particular engine to use or as a checklist for reviewing any other technology in a broad variety of organizations.

Chapter 1

Theoretical part

1.1 What is big data

1.1.1 How did the big data concept appear

Humanity has been naturally generating all kinds of information throughout its history – with getting to know the world, describing it and transferring knowledge to others. With the evolution of science data was becoming more and more diverse and stored in different formats with different techniques. From tally sticks through books and magnetic tape, people finally came to the point where information became digital. It was the beginning of a completely new era of storing and processing data.

Nowadays, the amount of information in the world is growing rapidly. For example, last year there was generated 16.1ZB of data, and according to IDC the global datasphere will grow to 163ZB by 2025[1].

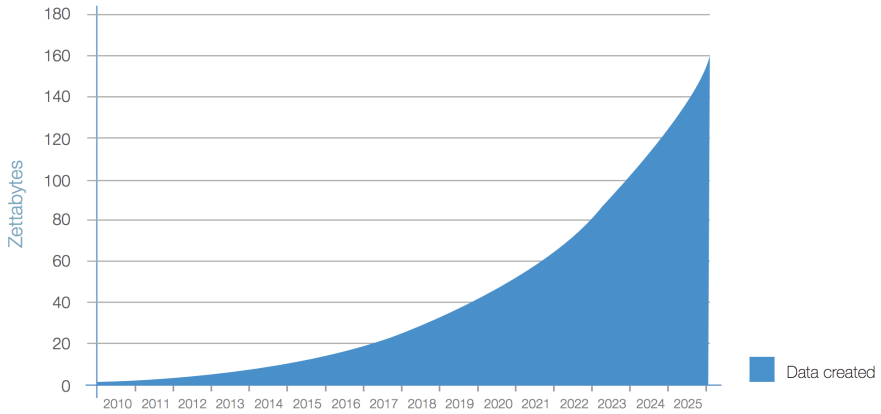


Figure 1.1: Annual size of the global datasphere[1]

But what does ”big data” exactly mean? Does it stand only for enormous amount of data or is there something more? The first appearance of term ”big data” refers to 1999, when it was used in the article ”Visually exploring gigabyte data sets in real time” published by the Association for Computing Machinery. Two years later, Doug Laney from Gartner published his paper ”3D Data Management: Controlling Data Volume, Velocity and Variety”, where he defines i.e. 3V (volume, velocity, variety) which will come to be the commonly-accepted characteristics of big data[2]. Those three attributes capture the main aspects determining if some particular data can be considered ”big” or not:

- the ”volume” characteristic defines the amount of data, which is usually ranges of petabytes.
- the ”velocity” characteristic is the measure of how fast the new data appears and are processed
- the ”variety” characteristic represents the diversity of data and an intention to caption all data related to the decision-making process, whichever structure they have (structured, unstructured or semi-structured)

Quite often in the context of big data, there is mentioned the fourth V standing for veracity. Veracity refers to the data quality and its trustworthiness. It is not that common as the classical 3V, but it is used more and more in the last years.

In essence, big data is a new way of looking on data and processing them along with a set of technologies necessary for this. Since 1970, when a concept of relational database was proposed by E.F.Codd, it was a standard approach to any manipulations with data (mostly storing them in the predefined model and querying with SQL).

But as data became more diverse and its amount increased significantly, the relational databases became insufficient for working with them effectively as they were pushed to its limits. It turned out that classic approaches and techniques can not be scaled to handle such loads. There are several reasons causing this.

First of all, relational databases were not designed for handling data in the range of petabytes; it would require i.e. vertical scaling which means adding either more CPU cores or memory. There are several problems with vertical scaling – it is expensive, not all vendors allow to do it, and it increases the risk of hardware failures which can cause major system outages. Second, relational databases were created primarily for storing of steady data, not for rapid growth; hence it faces certain problems with handling data generated with a high velocity (such is the most of big data). And finally, relational databases are able to work only with data tailored to the predefined model

(or schema); and since the vast majority of records in big data is either semi-structured or unstructured, it cannot be used with relational databases or must be reformatted accordingly; whatever of these would require a lot of resources or is completely impossible.

That was the moment when new technologies started breaking into the market to fill the gap created by users in need of working with changed data and lack of the tools capable to do it efficiently. The most important one, which is today de facto a synonym for big data, is NoSQL (this acronym stands for "Not Only SQL"). A NoSQL database concept was created as the opposite of the relational database model – it is designed to work with data without putting them into a model with tabular relationships. It was developed primarily in order to deal with limitations of SQL databases, especially scalability and multi-structured data.

1.1.2 NoSQL vs. RDBMS

To analyze these two database approaches, it is important to specify the comparison criteria. To start with, it is definitely worth mentioning the CAP theorem formulated by Eric Brewer. According to this theorem, it is impossible for a networked shared-data system to simultaneously provide more than two out of the following three guarantees[3]:

- consistency (meaning all nodes within the system have the same and up-to-date version of data)
- availability (guarantee that every request to the system will receive either success or failure response)
- partition tolerance (ensuring the ability of the system to be scaled horizontally)

The proven correctness of the CAP theorem implies a need for making trade-offs when choosing which of its aspects (and to what extent) will be provided in the certain system. And the way this problem is solved in relational and NoSQL databases differs significantly.

In the context of database technologies, data are consistent if they are completely the same across all existing instances at any moment of time. Database consistency refers to the requirement that any given database transaction must change affected data only in allowed ways[4] to ensure that any data written to the database are valid according to all rules defined in it (e.g. all integrity constraints are met).

Consistency model determines rules for visibility and apparent order of updates[5]. For decades the traditional approach to databases was based on ACID model, where A stands for availability, C for consistency, I for isolation and D for durability. This model's properties are mainly focused on providing

maximal possible consistency. But as the CAP theorem claims, once the consistency is ensured, it is possible to choose either availability or partition tolerance. Most of relational databases are provided with availability and since then they cannot be distributed across several nodes. Despite that, some of them can be configured in a way allowing horizontal scaling but restricting its availability.

But as new database technologies came, new consistency models appeared naturally. One of them, called BASE (Basically Available, Soft state, Eventual consistency) was created in the late 1990s to capture the emerging design approaches for high availability[6]. What it suggests is to be optimistic about the consistency by not forcing every operation to put the database in a consistent state necessarily[7]. By sacrificing this aspect, the BASE model allows the system to achieve a higher level of scalability which is unreachable within ACID model.

Undoubtedly, it is impossible to say that any of these approaches is better than another one. Another important point is to remember that it is not always possible to replace SQL with NoSQL and vice versa, which is implied by the different concepts lying in their basement coupled with consideration and adjustments in design. The decision to go either with SQL or NoSQL should be made based on a detailed analysis of the application architecture and its purposes along with use cases and possible limitations.

1.1.3 Big data architecture and workflow

As big data appears to be different from common one, its processing requires not only new technologies but also a special architectural design tailored to its specifics.

Each of big data 3V is thus reflected in different aspects of its processing. High volume implies a necessity of enabling powerful batch processing mechanisms for dealing with larger amount of data that have previously been saved to some storage and now need to be processed. For handling velocity data, the system should be capable of executing interactive queries to give a result based on all available information including the most recent pieces of it. Wide variety of data sets requires having an extensible storage system along with advanced techniques for accessing and then integrating them to the whole processing pipeline.

Big data architecture is a model of how big data and other information assets will be captured, stored, managed and made accessible to various user groups and applications[8]. In other words, it describes the way big data works as it flows through all components, from the raw data extracted from the data sources to the insights derived by end users using various analytical applications.

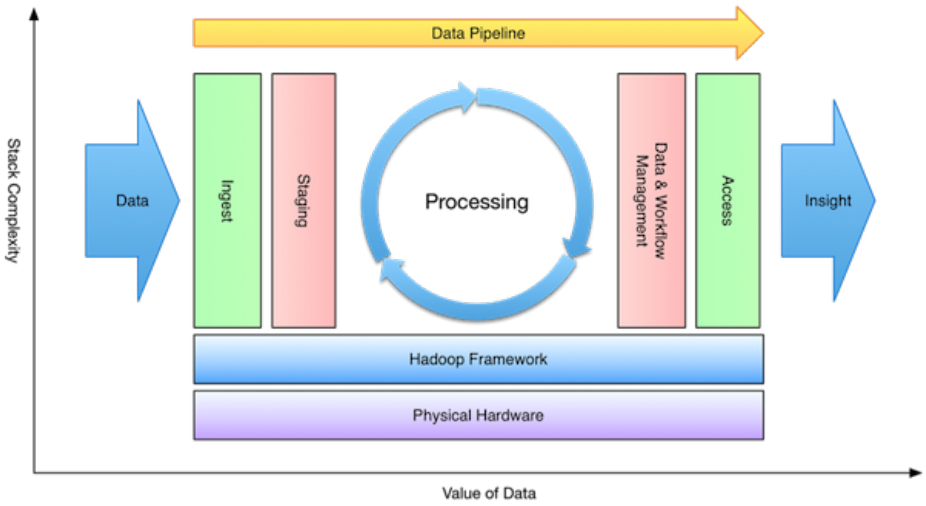


Figure 1.2: High-level view on big data architecture stages; although this schema includes Hadoop framework, the workflow would be similar for any other components.

Big data architectural design is based on a simple idea of connecting original data sources and business intelligence applications through data warehouses and analytic platforms integrated all together into a single system.

Of course, creating an architectural solution for big data system from the scratch is a task far from trivial. Nowadays, most of the models are built based on the same set of consideration concepts with slight differences depending on the technologies stack used for solving the particular task.

Any data analytics workflow is based on a simple sequence of actions done to data which can be simplistically described as acquire – prepare – analyze – report – act. Therefore, main components of analytical systems usually include:

- storage system such as a database or a file system
- ETL engine for preparing the data for being analyzed
- SQL engine for executing queries giving desired information
- BI tools for reporting and visual representation of gained insights

Although such design might seem simple, building and orchestrating the whole system brings certain challenges. First of all, it is essential to define the scope of processing: to determine desired outcome, understand which particular insights should be retrieved and find out which data should be analyzed for achieving this. Next step is choosing of components performing the best for defined goals. Nowadays market offers a wide variety of options

when it comes to data analytics software; however, it does not make the choice easier. At the same time, it is important to take into consideration not only the functionality of separate components but also its ability to be integrated with each other for creating a single pipeline. Also, the whole infrastructure should be optimized and adaptable for possible future changes. The reason is simple – if the current architecture will become insuffcient to solve new problems in a short period of time, the only solution is to change it which will necessarily entail multiple changes in the whole system. Needless to say, it is very diiňČcult (if not impossible) and expensive procedure, which should be avoided at almost any cost. Needless to say, it is very difficult (if not impossible) and expensive procedure, which is worth avoiding at almost any cost.

1.2 What is Data Analytics

1.2.1 Data vs. Knowledge

Of course, people were always interested in discovering what is hidden in their data since raw data by itself can not answer any questions or give somewhat useful insights. An idea of deriving meaningful information and its further usage is pictured by the DIKW (data, information, knowledge, wisdom) pyramid which represents the information hierarchy model.

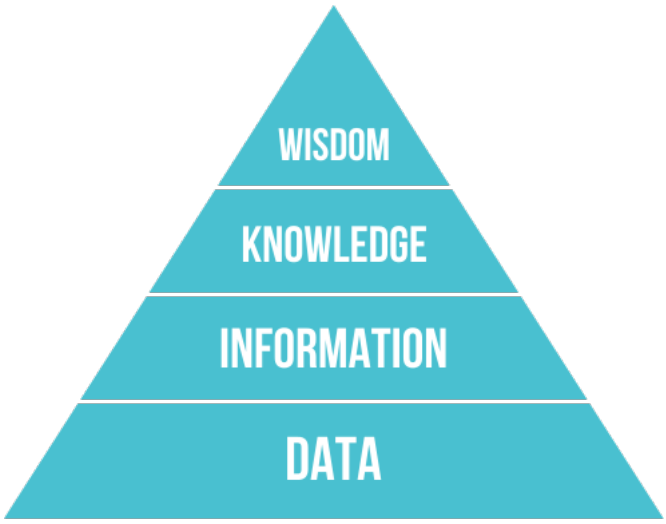


Figure 1.3: DIKW pyramid[10]

This model demonstrates the fact that any data gets its meaning after processing with a purpose. Despite data is at the bottom of the pyramid, it does not mean it is not important by itself. Data quality and accuracy are important because it is impossible to get correct and useful results based

on incomplete, inaccurate or imprecise data. Moreover, results obtained by dealing with invalid data can be considered useless or even harmful.

1.2.2 Data Analytics in general

Data analytics is a process of inspecting, cleansing, transforming, and modeling data with the goal of discovering useful information, suggesting conclusions, and supporting decision-making[11]. In general, the idea of data analytics is to derive meaningful insights from that data and to convert knowledge into action[12].

Data analytics includes a wide range of applications: from business intelligence (BI), reporting and online analytical processing (OLAP) to various forms of advanced analytics[13]. It is important to note that these applications do not only serve the purpose of direct data analysis, but also solve various tasks such as collecting, integrating and preparing the data. At the same time, they deal with development, testing and revising analytical models.

It would be a mistake to consider the whole data analytic field applicable only for big data; it is a general term for any type of processing that looks at historical data over time[14]. Today, data analytics is tightly coupled with the term big data mainly because it has evolved to being capable of working with data that can be considered ”big” (according to the definition given before). This evolution, in turn, was caused by growing market demand for systems that tolerate intense requirements of big data.

1.3 Big data analytics for business

As data analytics is used for solving many various tasks, it is often divided into four categories:

- prescriptive
- predictive
- diagnostic
- descriptive[15]

Each of these answers certain questions, and together they deliver complete insight helping to make more accurate business decisions.

Big data analytics plays a very important role for the business where it is implemented. By itself it is only able to provide with certain insights and forecasts derived from the raw data. When it comes directly to the decision-making process, there is a need of using some additional tools, which are usually covered by an umbrella term of business intelligence (often referred as BI). Since most of businesses nowadays want to have implemented both

data analytics and business intelligence (in order to get the most precise overview possible) and the market reacts to these needs, these two terms became interchangeable.

The main reason why BI tools are not coupled with data analytics technologies into the single term and framework is that the whole process of harnessing data is executed by users with different scope of knowledge required in their job positions. Wayne Eckerson in his report "Big Data Analytics: Profiling the Use of Analytical Platforms in User Organizations" defines two groups of users:

- casual users (executives, managers, front-line workers) who primarily use reports and dashboards that deliver answers to predefined questions
- power users (business analysts, analytical modelers and data scientists) who perform ad hoc queries against a variety of sources

BI tools are therefore designed primarily for casual users, providing interfaces for indirect work with data based on extracted insight. Data analytics tools are used by power users and give them more freedom in interacting with raw data and accessing all platform components from the inside. These tools are not interchangeable since they interact with data in different manners and are insufficient for solving each other's problems.

Natural evolution of BI tools gave birth to its new generation called self-service BI. The main feature of self-service BI tools is that they can be used by users with limited technical skillset. Together with the ability to provide access to the information stored anywhere it brings a significant value to the business by accelerated acting based on new insights. It also allows new users to start working with the information as soon as possible, without a need to study the tools by itself or getting familiar with the whole data analytics infrastructure. Another important point is that self-service BI tool decreases the IT departments workload – casual users don't need to ask them about providing data access/permissions/technologies etc.

As the self-service BI appearance puts a human face on a data analysis usage, more and more companies tend to transform itself into more data-driven by adopting a data analytics culture. It would be a mistake to consider such transformation efficient only for bigger companies like corporations and enterprises, about which Forbes claims that "data analytics is no longer a nice option – it's the core of the enterprise"[16]. Implemented correctly, data analytics integration brings same benefits to all companies regardless of its size. Of course, it is not a silver bullet and requires certain resources but taking into consideration the trend of being data-driven can become a must in the following years.

Chapter 2

Practical part

The previous chapter describes the general concept of big data, its influence on data analytics and their role in a business field; it is mostly focused on reviewing existing approaches and architectural patterns.

The practical part of this work includes several points. First of all, it describes chosen SQL engines used in big data analytics based on its documentation and best practices. Then it defines criteria that should be taken into consideration while choosing a particular engine to use within an analytic system. Using these criteria, the engines are compared which allows to determine the best use cases for each of them.

2.1 List of engines to compare

Nowadays, there are dozens of engines available on the market. At first glance most of them may seem very similar which definitely does not make a choice easier. Picking up the right technology is extremely important as it defines not only the result achieved by its adoption but also the amount of effort putted in both setting the whole thing up and then maintaining it.

I have chosen the following engines for consideration within this work:

- Apache Hive
- Apache Impala
- Apache Drill
- Presto

This choice is based on an aspiration to review the most well-known technologies (though it is a debatable aspect) which are widely adopted by various organizations of different scale. Prevalence of these technologies also indicates

that they are mature enough to be considered stable and trustworthy which ensures they will not become outdated anytime soon.

All of the engines reviewed within this research are open-source projects which might seem insufficient as there are a lot of commercial solutions provided by such well-known companies as Teradata, HP, Microsoft etc. The reason is that this paper is supposed to help with choosing a particular SQL engine for designing a new custom data analytics infrastructure or to integrate it into an already existing one while commercial solutions are mostly sold as a whole and do not allow using its particular components separately. Those solutions are more suitable for enterprise-like organizations as a vendor takes care about designing, setting up and maintaining the platform according to specific customer needs; however, purchasing it is usually quite expensive so it is not affordable for smaller businesses. Also, such preconfigured solutions cannot be modified easily and usually require consulting from vendor side.

2.2 Overview of separate engines

2.2.1 Apache Hive

2.2.1.1 Overview

The Apache Hive data warehouse software facilitates reading, writing, and managing large datasets residing in distributed storage using SQL [17]. tommorow

2.2.1.2 HiveQL

For quering data Hive uses HiveQL – an SQL dialect supporting both DDL and DML statements[29]. HiveQL DDL statements allow users to define various data units standard for Hive such as database, table, partition or bucket and thereby create a schema structure. In addition, HiveQL DDL includes statements for using SerDes[30] (see section 2.2.1.4) for creating schemas while reading the data instead of doing it manually and then reformatting the data to fit it. HiveQL DML statements then allow to fill schemas with data by loading them from files or even intermediate queries results, access and modify data in tables, write query results into filesystem directories and merging tables[32].

HiveQL has some subtle functionality differences from classic SQL and does not fully meet the SQL-92 standard, but at the same time provides additional extensions. One of them is support of CREATE TABLE AS SELECT (CTAS) statement helping quickly derive Hive tables from other tables[31]. Another useful feature is Multi Table Insert – statement for inserting data into multiple tables within a single operation[32]. The main benefit here is that using this statement Hive scans data and applies query operators on them only once which makes execution faster than sequential execution of inserting data into tables with the traditional INSERT statement.

Probably one of the most powerful HiveQL features is support of user defined functions (UDF). It allows using functions written in Java or Scala inside of HiveQL statements. Main benefits of using UDF are:

- any function needed for specific part of work can be implemented
- easy to share, reusable code
- easy refactoring
- make statements more readable

Using UDF is very simple. For doing this, it is needed to build function source code, add the resulting jarfile into Hive and then create temporary function by running homonymous Hive command and specifying the function name as it will be used in HiveQL statements and a fully qualified package name.

On the other hand, as any other language HiveQL has certain limitations and disadvantages; nevertheless, most of them are caused only by currently immature support of specific aspects (not by the technical impracticability) and will be possibly fixed in future releases.

To name a few, Hive generally supports subqueries only in the FROM clause (and some types of them in the WHERE clause, but not in all cases)[33]. Also, Hive provides quite limited support of transactions. First of all, enabling transactions support requires certain configuration of both client and server sides. Second, executing transaction is only possible for data stored in ORC file format. Finally, all language operations are auto-commit – BEGIN, COMMIT, and ROLLBACK are not yet supported[34].

2.2.1.3 Architecture

The figure 2.2.1.3 illustrates Hive architecture.

The block called Driver represents the query engine which is responsible for running HiveQL queries. Queries then are converted into MapReduce, Tez or Spark jobs and then the platform is referred accordingly as a "classic" Hive (running MapReduce jobs), Hive on Spark or Hive on Tez. MapReduce, Tez and Spark are frameworks for processing data based on different programming paradigms. All of them have been built with different purposes and therefore provide various features improving user experience for different use cases. Any of these frameworks can be choosen for running jobs which gives high flexibility and allows to tune the whole tool according to specific needs.

Thrift server (also referenced as HiveServer2 or HS2) is a server interface based on Thrift RPC that enables remote clients to execute queries against Hive[26]. Among others, it supports JDBC and ODBC APIs allowing connecting a wide range of applications and tools to Hive and leverage the whole platform even more efficiently.

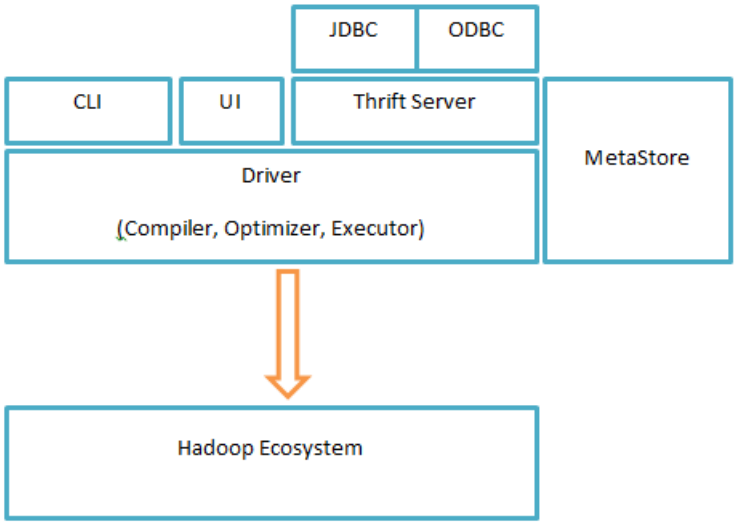


Figure 2.1: Hive architecture[35]

Next component, metastore, is in charge of storing all metadata necessary for manipulating with data. This includes information about tables and partitions structure (column names, data types etc.), serdes used for reaiding and writing particular data, HDFS files where data are stored and so on. Metadata are obtained in the moment of table creation and then used every time its data are accessed.

Speaking about datasources Hive is able to work with, it is necessary to take into consideration the fact that it is built on top of Apache Hadoop – framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models [18]. Hadoop includes its own file system called HDFS (Hadoop Distributed File System) and that is the only data source Hive works with. It might be a problem if the data for processing are stored at some different storage – in order to make Hive process them, it would be necessary to transfer them to a HDFS cluster first in this case.

2.2.1.4 Key features

One of Hive's greatest assets is its ability to interpret unstructured or semi-structured data by transforming them into Hive tables. Using the schema-on-read concept allows avoiding setting up ETL tools and workflows, which is usually not only difficult and costly, but also makes the whole data processing pipeline slower and more cumbersome.

Inside the Hive platform, this concept is implemented with a component called SerDe (which stands for Serializer and Deserializer) – interface that tells Hive how it should translate the data into something that Hive can process[27], e.g. to Hive tables. In other words, SerDe helps Hive to interpret loaded data and makes it able to query them. Now there are a lot of SerDes for handling all kinds of data. Out of box Hive has several built-in SerDes for such popular data formats as CSV, JSON, Avro, Parquet etc. What is probably more important is the possibility of creating custom SerDe for any other data format as any SerDe is basically just a Java class implementing the general interface[28]. Besides, there are a lot of SerDes written by third-party developers and available as open-source projects, so in many cases it is enough to simply find a suitable one and leverage it.

In addition to UDF (see section 2.2.1.2), Hive allows creating even more flexible and robust statements with a tool called HPL/SQL – Hive Hybrid Procedural SQL On Hadoop[36]. This tool implements PL/SQL – procedural extension for SQL created by Oracle and allows writing HiveQL statements using procedural language elements (such as variables, loops, conditions etc.).

As mentioned at section 2.2.1.3, Hive can on multiple backend platforms, such as MapReduce, Tez and Spark. But what is the difference and why it is beneficial for the whole product?

Initially Hive was running using MapReduce as at that time it was the most suitable options for fulfilling Hive needs. But as technologies have been developing, various data processing frameworks were evolving as well. One of them was Tez which appeared to be more effective, powerful and flexible than MapReduce, so it was decided to support it as an alternative backend platform in order to increase overall performance of Hive. Another popular cluster-computing framework, Spark, was adopted primarily because of its popularity among users. It allows more organizations using Hive without need to change their existing infrastructure.

Whilst Hive have been integrating new backend platforms, it still supports all three at the same time, so users can choose which one suits the most for their needs. It helps to increase Hive's adoption among a wide variety of different companies which is also beneficial for the platform itself as it is open-source and bigger number of users means bigger number of potential contributors.

2.2.2 Apache Impala

2.2.2.1 Overview

Apache Impala is the open source, native analytic database for Apache Hadoop, providing low latency and high concurrency for BI and analytic queries (not delivered by batch frameworks such as Apache Hive)[19]. Currently Impala is an Apache Software Foundation project and is available under Apache

licence version 2.0. Also, Impala is now available as a part of CDH – advanced system provided by Cloudera built from open source components enabling performing end-to-end Big Data workflows in enterprises.

Impala fully supports ANSI SQL and is compatible with a lot of popular BI tools (as it was originally targeted), which ensures seamless integration into already existing data analytics infrastructure. As noticed in Cloudera Engineering Blog, Impala has unlocked the ability to use business intelligence (BI) applications on Hadoop[20].

2.2.2.2 Architecture

Impala's distributed architecture is based on daemons processes responsible for all steps of query execution, such as accepting queries from client processes, orchestrating their execution across the cluster, and for executing individual query fragments[21] [22]. Impala runs on multiple machines in Hadoop clusters: each demon process is represented by the *impalad* process running on each of cluster's nodes. Incoming queries can be sent by impala-shell command, Hue, JDBC, or ODBC.

All Impala daemons are able to execute any of operations (e.g. they are not separated by functionality), which ensures fault-tolerance and provides load-balancing. When one or more hosts are down, Impala reroutes future queries to only use the available hosts, and Impala detects when the hosts come back up and begins using them again[37]. However, Impala does not provide any mechanism for assuring query fault tolerance. As intermediate data are kept only in the memory and are not written on a disk, in the case of failure at any execution point the whole query will fail. To get a result, the failed queries should be retrieved manually as Impala does not provide monitoring allowing to detect failed quires and rerun them automatically.

For storing metadata (information about schema objects) Impala uses Hive metastore which implies the necessity of installing and configuring it along with Impala itself[38]. Essentially, it is a simple MySQL or PostgreSQL database with a corresponding service running for connections handling. Also, part of metadata is kept in the node cache providing faster access to those related to the most recently used objects.

2.2.2.3 Key features

Impala's distributed architecture brings several important benefits. Probably one of the most significant characteristics of Impala is its performance. The reason is that using daemons allows to bypass converting incoming queries to MapReduce jobs which makes its execution much faster. Moreover, as daemons are running directly on data nodes, data can be accessed with a lightning speed. Multiple tests performed by Cloudera in 2016 and 2017 show that Impala achieved better results

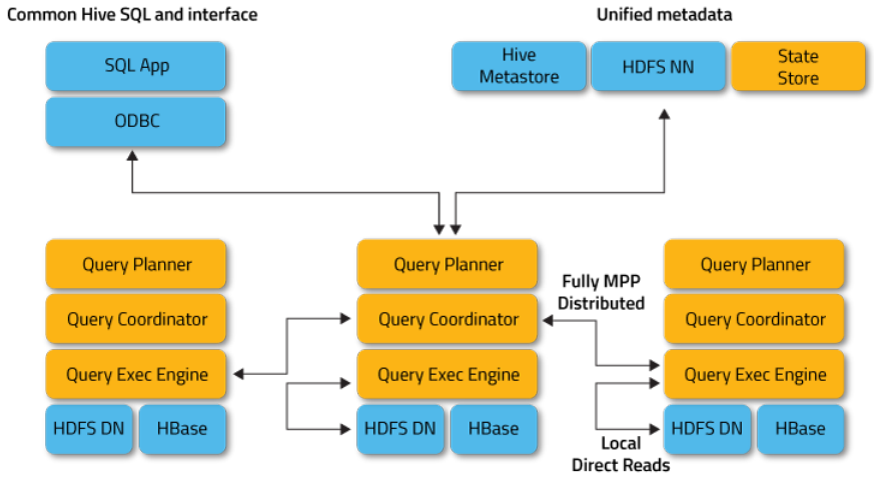


Figure 2.2: Impala architecture[39]

- compared to batch and procedural development engines (such as Hive-on-Tez and Spark SQL 1.5)[23]
- compared to analytic databases (Greenplum) and SQL-on-Hadoop engines (Spark SQL 2.1, Presto 0.160, and Hive 2.1 with LLAP from HDP 2.5)[24]

Tests were performed with using TPC-DS – performance benchmark providing representative evaluation of performance as a general purpose decision support system[25] which ensures trustworthiness of obtained results (though tests organizer was Cloudera who can be called ”patron” of Impala). Also, Impala can be tuned for achieving even better performance with using more advanced methods such as specific data partitioning, optimizing queries itself and gathering schema statistics.

Another benefit of distributing queries across a cluster of computing nodes is the possibility to scale the whole system easily just by putting new nodes into rotation. The simple concept of horizontal scaling works pretty well with Impala – if disk throughput is insufficient or memory limits affect the execution speed, then it is enough to add more machines into the cluster and run a daemon on it to solve the problem. Also, this approach is really cost-effective when it comes to hardware resources usage as a cluster usually consists from relatively cheap units. The whole system then could be scaled up without buying any very expensive appliances or scaled down by removing machines from it to spend less on its maintenance.

But as an amount of memory is physically limited for each node, memory-intensive operations can run out of this limit failing the whole operation.

However, Impala is still able to successfully execute such queries with setting up disk spilling. This principle allows writing intermediate data on the disk instead of keeping them in memory. Anyways, this option should be used only in particular cases where a query cannot be optimized to fit the limit or the limit can not be extended. The reason is that spilling on disk significantly slows processing down due to constant writing to and reading from a disk.

To optimize memory usage, Impala also features HDFS caching – technology allowing to define which tables or partitions should be kept in memory no matter if they are currently processed or not. This technique is especially useful for smaller, frequently accessed data.

Impala is capable to access data stored on HDFS, Amazon S3 and HBase.

HDFS is a primary data storage medium used by Impala. By default, all the data files for a database, table, or partition are located within nested folders within the HDFS file system[40]. An important benefit of using HDFS is that it provides data redundancy protecting the system from hardware and network problems causing unavailability of certain nodes. Furthermore, using HDFS is practical because in this case there is no need of transforming or moving the data as queries are executed directly on datanodes.

Amazon S3 is almost fully supported by Impala starting from version 2.6. Impala supports both queries and DML for data residing on Amazon S3 but it can not be used as the only filesystem in the cluster, because Impala requires that the default filesystem for the cluster be HDFS[41]. Also, as noticed in the documentation, queries against S3 data are less performant, so S3 is more suitable for keeping the data which are queried occasionally rather than those actively used which are supposed to be kept in default HDFS.

Unlike HDFS and S3, HBase is typically used for more specific use cases rather than simple storing common data. It is mostly used as an additional storage to the default Impala HDFS or for storing data which are specific at some way and are handled by HBase more effectively because of its design (for example, tables with many – hundreds to thousands – columns representing subject's attributes or data which changes very quickly, such as counters).

Impala was initially designed to work with HDFS therefore it obviously supports designated Hadoop file formats: Avro, RCFile, SequenceFile, Parquet and plain text; it can proceed compressed data as well. However, even after the list of supported data platforms was extended, Impala still is not capable of working with further file formats.

In order to query data, Impala needs to have some predefined model describing them; it ususally consist of such schema objects as databases, tables and partitions. As Impala was initially designed to work with HDFS, the folder structure of HDFS was taken as a concept for creating a data model. Database is a logical container for a group of tables and each one is physically represented by a directory in HDFS[42]. Tables contain the actual data written into classic row and column structure. Each table has an associated file format. If the table uses HDFS, it is associated with a particular directory

and contains all data underneath that directory[43]. In general, there is no need in complex schema modeling; nevertheless, some optimization can be achieved with subtle changes in the data model.

2.2.3 Apache Drill

2.2.3.1 Overview

Drill is an Apache open-source SQL query engine for Big Data exploration[44]. It was primarily designed for high performance on large datasets along with ability to query it on a high speed necessary for integration with BI environments[45]. Drill is often referenced as an attempt to build an open source version of Google Dremel[46] what was admitted in its proposal to Apache incubator[47]. From early on, Drill development was strongly impacted by its users and developers together with business community which had influenced many of its facets. Jacques Nadeau, a leading architect of Drill, pointed out that one reason Drill can manage this combination of flexibility and performance is that it was designed from the start to have these capabilities[48].

2.2.3.2 Architecture

Drill features distributed execution environment[49] capable of scaling from a single node to thousands of servers. The whole engine consist of multiple instances of service called Drillbit responsible for accepting requests from the client, processing the queries, and returning results to the client[49]. All Drillbits are equal, there is no master-slave concept – every node is self-sufficient and is able to accept and run queries obtained from a client. Drill can be installed on any distributed cluster environment; the only pre-requirement for running it is to have installed Zookeeper. ZooKeeper is a centralized service for maintaining configuration information and providing distributed synchronization[50]. Inside a Drill cluster, Zookeeper is used for monitoring the cluster status and discovering available Drillbits before submitting queries[51]. Even though clients can communicate with a specific Drillbit directly, it is not recommended; Zookeeper is integrated into the system to simplify the load distribution between nodes and prevent clients of having troubles while connecting to nodes directly.

Under a Drillbit hood are hidden its three main components:

- RPC endpoint
- SQL parser
- storage plugin interface.

RPC endpoint serves to communicate with clients using remote procedure call protocol. It allows to execute a procedure or a function on a machine

different from the one it was called from without explicit specifying of call details (e.g. in the same way as if the procedure is called on the same machine).

For parsing incoming SQL queries Drillbits have integrated Apache Calcite – open-source SQL parser and validator[51].

Storage plugins are connectors used by Drillbits to access data which should be queried. Different plugins can connect to various data sources, such as a database, a file on a local or distributed file system, or a Hive metastore[52]. They allow Drillbits to execute read and write operations and provides access to source metadata. They are highly configurable and easy to use: multiple plugins can be installed at the same time, enabled or disabled and configured separately. Information about any plugin updates is broadcasted to all Drill nodes, so there is no need to restart any of cluster nodes after making any changes.

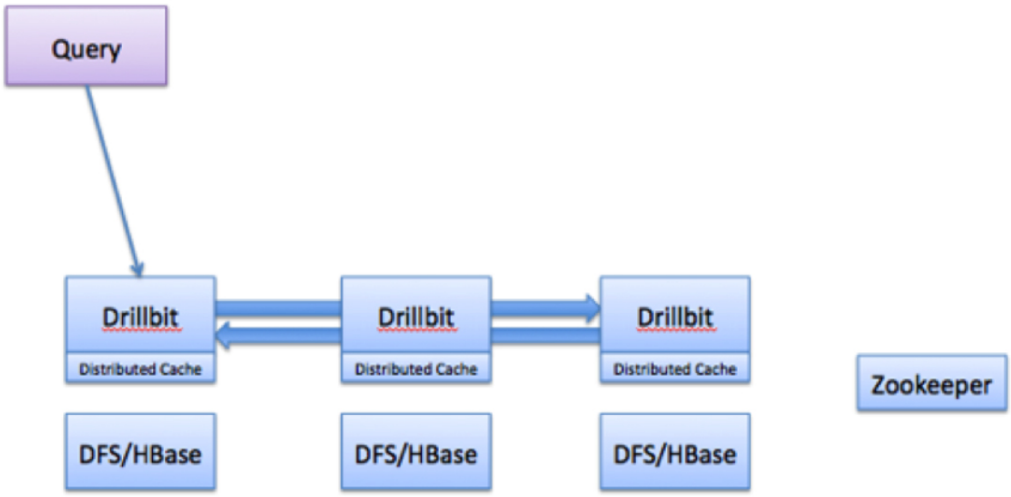


Figure 2.3: Drill query flow[45]

Drill's architecture is based on massive parallel processing concept where a large amount of separate computers works in parallel. It can also be described as i.e. shared-nothing architecture[55] which means that every node is independent from others and is able to work by its own; none of resources (such as RAM, disk space etc.) are shared between nodes in a cluster. At the same time Drill has one component common for all nodes – distributed metadata cache. It is an auxiliary component used for managing metadata and configuration information across various nodes[45] thus increasing overall engine performance.

All operations are executed in node memory and none of intermediate results are not written into disk. It helps to reduce overall execution time and disk load, but on the other hand requires more memory to successfully execute

all incoming queries. The only case when Drill persists data to disk is when a memory limit is exceeded and some resources must be freed immediately to allow the engine continue working.

2.2.3.3 Key features

Probably the most significant feature of Drill is that it does not required any data pre-processing such as ETL operations and creating schemas for querying them. It makes using Drill easier (especially for business users) by several reasons:

- no need to structure data before querying
- no need to create and manage metadata repository
- historical data can be queried in the same way as actual one

Raw plain text files can be processed without converting them to any specified format or putting its content into tables or structures. Drill does not need to know data structure before starting processing it as it is able to discover a data schema on-the-fly regardless of an input format. A query is automatically compiled and re-compiled during the execution phase, based on the actual data flowing through the system[53]. Due to this mechanism Drill can process even data with evolving schema or schema-less data (such as MongoDB or raw files). However, Drill is not about denying data schemas at all – when it comes to handling data with known predefined schema (for example, NoSQL database tables, Avro, Parquet etc.), it is leveraged during a querying process.

Drill features a JSON data model[54] which allows querying both fixed-schema and schema-free data formats at the same time. Inside of the engine, all input data regardless of its format are transformed to a JSON structure which standartizes the following process of querying them. It distinguishes Drill from most of traditional query engines on the market relying on a rigid relational data models. The main benefit of this approach is that it allows the engine to process both schema-fixed and schema-free formats and does not make any difference between processing flat and nested records.

Because of Drill's distributed architecture, Drillbits can be run on any machine in a cluster – including those storing the data. It allows leveraging i.e. data locality – ability to process the data directly where it resides without moving it over the network. It significantly reduces latency and the risk of data loss thus improving engine's reliability and end user experience.

Drill adopts an optimistic execution model to process queries[45]. Queries execution is meant to be lightweight and fast process with as few obstacles and dependencies as possible: in-memory execution eliminates errors while accessing the disk, data locality prevents issues with transporting the data over the network etc. Therefore, Drill considers failures appearance to be

highly unlikely. However, on the node level Drill is able to handle failures and rerun the failed query.

2.2.4 Presto

2.2.4.1 Overview

Presto is an open source distributed SQL query engine for running interactive analytic queries against data sources of all sizes[56]. The idea behind is to allow quick, high-performance processing of data no matter where they are stored and at the same time avoid complex ETL operations before. This engine was designed for ad-hoc quering data in real time, so typically a query is executed in a range of minutes.

Developed by Facebook, Presto was successfully adopted in many companies and received many positive references. In 2015, Teradata started actively contributing to Presto and later became the first commercial vendor to offer enterprise support for it[57]. This has affected Presto a lot as more and more organizations (including bigger ones) started using, and, consequently, contributing by either fixing the most obstructing bugs or implementing missing features. Needless to say, different organizations bring up changes which are the most relevant for them developing Presto in various ways. But altogether all these changes allow the whole project to evolve, grow and become more mature.

2.2.4.2 Architecture

Figure 2.2.4.2 describes a basic idea of Presto architecture.

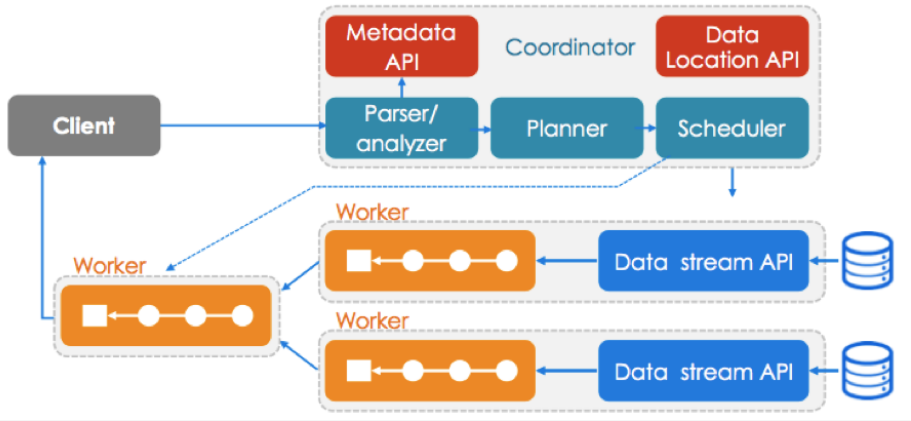


Figure 2.4: Presto architecture[58]

The concept of Presto as a distributed query engine is to assign different types of work to different nodes – single instances of Presto installed on

machines. There are two types of nodes: a coordinator and workers. Coordinator can be considered as a control center of the whole engine: it parses query statements, plans the query running and manages workers. To plan queries, coordinator creates a logical model and translates it into separate tasks which are then transmitted to workers. Workers, in turn, execute tasks obtained from the coordinator and process data accordingly by fetching it from connectors and exchange intermediate data with each other[59]. The coordinator is also responsible for all communication between the engine and the client in both ways: it obtains queries to execute and fetches the result from workers to return it to the client.

Connector is another important part of Presto. In essence, it is an alternative for database drivers (such as JDBC or ODBC) allowing Presto to interact with a resource using a standard API[59]. Out-of-the-box, Presto contains several basic connectors, but there are also plenty of third-party developers providing connectors for connecting to a broad variety of different datasources (such as Cassandra, Kafka, MongoDB, MySQL etc.).

Every connector is mounted in a separate catalog – a structure keeping information about schemas and tables. Catalog references data sources via connectors[59]. In order to query a table from a specified schema, both the table and the schema should be defined in the corresponding catalog. Catalogs are created by adding properties files into the specified system folder. Every SQL statement is run against one or several catalogs allowing user to query various data sources with a single SQL statement.

Another component essential for querying data is Hive metastore. Presto uses it in the same way as Hive – for accessing metadata stores in separate storage through a service exposed by Hive metastore.

Presto's architecture can be described as in-memory which means that all queries are executed in memory. It significantly increases execution speed as intermediate results are not written on disk, so the engine does not spend time on additional I/O operations during execution. On the other hand, it affects engine's ability to handle queries which need more memory than a node has available. Such queries are usually killed by Presto with an appropriate exception risen. Such approach enables more effective memory allocation for a big number of smaller tasks but in order to allow execution of bigger ones it is required to configure spilling to disk. It is an advanced technology allowing to use a disk as storage for intermediate data to overcome the memory limit. However, spill-to-disk is not a silver bullet: there is still a chance of failure during dividing intermediate data or loading it from the disk; at the same time, not all operations support spilling to disk, and each handles spilling differently[61].

2.2.4.3 Key features

Presto is able to run queries combining data from multiple different data storage at the same time. It only requires having corresponding connectors configured for all data sources which should be queried and Presto will do the rest. This allows running analytic processes across all data available, no matter where and how they are stored which might be especially beneficial in the case of multiple datastores used in the same organization.

Teradata was participating in Presto's development not only by contributing to its codebase. In 2016, Teradata announced the certification of multiple business intelligence (BI) and visualization solutions on the Teradata Distribution of Presto[62]. Practically it means that Presto can be seamlessly integrated to various infrastructures which use these BI tools and thus leveraged in a different data analytics pipelines. It implies the fact that more and more companies are able to use Presto (or at least to give it a try) without a need to rebuild already existing infrastructure. And same as for any open-source project, growing community makes bigger impact and helps to develop more mature and tuned product.

Presto queries are written in ANSI-compatible SQL. It ensures seamless adopting Presto into already existing data analytic infrastructures as there is no need to transform already existing queries. At the same time, it makes Presto usage easier for analysts and developers as it does not require learning any additional language or specific SQL dialect for working with it.

Presto supports multiple file formats, mostly Hadoop ones: plain text, SequenceFile, RCFile, ORC and Parquet[63]. Also, Presto features supporting multiple compression codecs such as Snappy, LZ4 and gzip.

The only thing needed for starting using Presto is to set up a cluster and configure its nodes. The process is quite straightforward and does not require any complex orchestration or multistage deployment – it it enough to install Presto package on machines, modify the configuration files, and then simply start Presto with a single command. At the same time, Presto is not exacting when it comes to hardware requirements; a cluster may be built with commodity servers or even in a virtualized environment. Another options (which might be even more relevant nowadays) is to run a Presto in the cloud using Amazon Web Services. Amazon EMR, a managed Hadoop framework for processing data across dynamically scalable Amazon EC2 instances[64], allows not to care about capacity stats, scaling, monitoring or securing a cluster and can be set up swiftly.

Presto is highly configurable: the whole engine can be tuned to perform its best even in cases of specific tasks or problems appearing. It is possible to specify the amount of JVM memory reserved for accounting purposes, allow memory spilling, set a number of threads used by exchange clients to fetch data from other Presto nodes, select a library for regular expression functions and many others[60]. Even greater flexibility is achieved by ability to configure

every node separately and independently from other nodes.

Speaking about fault tolerance, Presto sacrifices it to have faster query processing as it was primarily designed for running short interactive queries. Therefore, if a process fails while processing, the whole query must be re-run which requires re-reading data from the data source and re-computing all intermediate data. On the other hand, Presto's execution speed is high enough to allow re-running failed queries without any major decrease in overall performance.

As regarding user interfaces, Presto can be controlled and operated via either a command-line interface or some of GUI consoles (mostly web-based). Presto CLI provides a terminal-based interactive shell for running queries[65]. If there is a need to have more user-friendly interface or to provide a wider or more sophisticated functionality, it is worth considering deploying some web console in addition to the built-it CLI. Presto has its own solution in a form of web interface for monitoring and managing queries[66]. After spinning it up on some HTTP port and opening the corresponding URL in a browser, the console displays a list of queries with its basic information, such as ID, state, percentage completed etc. A more detailed information about a single query can be obtained by clicking on its ID, which leads to a page containing graphical representation of various stages of the query and a list of tasks[66]. Also, this page has a button for killing currently running query which is especially useful for solving problems with blocked queries. In addition to the built-in web console, there are several GUI components designed by third-party developers. They have very different features (from advanced query editors to proxy servers), which makes them a good alternative to the default one.

Another way or running queries is using Presto libraries for different programming languages. It allows to start query execution directly from the source code and is used mostly in some automated scripts or more complicated applications using Presto as one of its components. It gives a lot of opportunities to harness Presto by integrating it in more complex data processing workflows or to already existing infrastructures.

2.3 Final report

2.3.1 Comparison criteria

When choosing any software component, it is important to review and evaluate different aspects of all technologies to assess its capabilities and understand whether it is suitable for a particular use case. At the same time, as SQL engines are meant to be a part of a complex composite system, it is important to take into account not only its own feature but also the way other infrastructure components are implemented to avoid integration issues.

2.3.1.1 Deployment options

Nowadays there are two main models for application deployment based on different computing solutions: on-premise and cloud.

On-premise means that the whole infrastructure is hosted and controlled by an organization using it. It includes both setting up and maintaining hardware appliance, infrastructure, network, operating system etc. This model requires both material and time initial investments, as it is necessary not only to purchase component itself but also spend some time and put an effort into making all of them work together. At the same time, on-premise infrastructure can be tailored to particular needs and gives total control over entire system; moreover, separate parts of an already existed system can be modified with changing requirements. However, the responsibility for seamless usage might be challenging it is necessary to ensure that a system is stable and robust enough to handle the load.

Cloud computing enables leveraging already existing system resources and preconfigured services hosted by a third-party provider. Instead of setting up own cluster and managing all the dependencies, users simply connect to a cloud platform, request for creating new instances and within minutes have them up and running and vice versa: if for some reasons too much resources were acquired, unused instances can be effortlessly shut down. It is possible to provision the exact amount of resources that is needed at the moment and then scale up or down easily as needs are changing. Cloud computing helps to optimize operational costs as it charges only for the resources which were actually consumed. A service provider is also in charge of ensuring resources availability and seamless user experience. Nevertheless, using cloud computing services implies certain restrictions and has its drawbacks. As all resources are fully owned and controlled by a provider, customization options are very limited therefore configuration set by him cannot be modified significantly. Another thing is that to be processed on a third-party provided appliance, data must be accessible for him which can cause certain concerns when it comes to sensitive or confidential information.

It's also important to consider how the rest of infrastucture components are deployed; if some of them is already running in the cloud it may make sense to bring the rest to the cloud as well.

2.3.1.2 Support provided

Most of big data software products are supposed to be used and maintained by persons having certain skillset and experience. It is caused not by its design, but the overall complexity of tasks solved with them. Naturally any organization sooner or later runs across some problems while operating a system, especially a complex one. Solving them is time-consuming and forces to switch the focus to troubleshoot a system instead of actually leveraging it.

Companies selling commercial solutions usually provides support and helps customer to deal with any issues occuring during the software exploitation. However, when it comes to open-source products, the situation is different: as there is no single vendor supplying it, support can not be provided by the developer company (as there is not any). Nevertheless, open-source products can be also supported but in a slightly different manner. Some software companies (for example, Cloudera and Hortonworks) offer complex platforms consisted of multiple open-source components with additional enhancements. These solutions or its separate parts could be purchased for free; however, it is not about them put to the market by itself but paid subscription for support provided for them. Moreover, such solutions are mostly e.g. enterprise-ready which means they are suitable for being used in larger organizations and fully meet their needs.

Getting professional support from third party vendors allows companies avoid wasting human resources on operating and troubleshooting infrastructure and focus on leveraging it instead. It makes perfect sense, especially for more complex and intricate systems where issues naturally occur oftentimes, can be difficult to solve and require a certain level of skills and experience from specialists working on them. After all, tools should serve people's needs and if it requires inexpedient amount of effort put to make it work it probably does not worth it and alternative options for implementing an infrastructure should be considered.

2.3.1.3 Supported data sources and file formats

Engine's capability of accessing and processing data stored in a certain way is one of the most important consideration in the designing a data analytics infrastructure. Every engine is optimized for processing particular data formats and able to access particular data storages.

Probably the key factor determining a list of engines to choose from is a kind of data needs to be analyzed. Is it some already existing, historical data or a workflow will start processing only new data coming once the whole thing is set? Most of SQL engines are optimized to perform the best working with certain file formats and are able to access specific datastores. For historical data it should be decided either to search for an engine capable of working efficiently with existing data storage set-up or consider transforming it in a way allowing data to be processed by a particular suitable engine. If currently there are multiple different datastores used it might be reasonable to standardize and centralize them either by moving all data to a single storage or with implementing data staging. Although it would require some additional work (amount of which may vary depending on solution complexity), it allows leveraging more engines, not only those capable of querying multiple resources at the same time.

However, if a pipeline is being implemented from the scratch and there are no prerequisites given by existing datasets, an engine can be chosen based foremost on its performance and features provided and then set a data storage infrastructure accordingly; the same idea applies for data formats.

In the end it is about tailoring the infrastructure for data or vice versa, so trade-offs are inevitable.

2.3.1.4 System requirements

As any other software, SQL engines have specific hardware and software requirements. Although most of them are more like a guideline rather than necessity, fulfilling them helps to operate the entire infrastructure in a more effective way and optimize cooperative usage of hardware and software.

Hardware requirements usually specify such system parameters as CPU performance, amount of RAM available, and storage capacity. Software requirements define an operating system and necessary software components installed, such as particular runtimes, drivers etc.

Unlike hardware requirements, software ones are mostly absolutely necessary to fulfill as without it an engine will not be able to run at all while unsuitable hardware would most probably just make it work unstably. At the same time, it is easier to tailor a system to have software requirements fulfilled as it can be done by installing missing components (which is usually a trivial task) but changing hardware configuration is more complex operation. At some cases system requirements, especially hardware ones can be satisfied only with on-premise deployment as not all cloud resources can be tailored according to them.

2.3.1.5 Customization options

SQL engines are meant to solve a wide variety of tasks; however, it is not always possible to do it leveraging only out-of-the-box features. For handling more sophisticated use cases some engines provide various possibilities for enhancing or expanding its functionality. The most common customization aspect is file formats or data storages that can be queried with this engine. For additional file formats support there is an option to implement custom SerDe (serializator/deserializator) telling an engine how the particular format should be read and parsed. Likewise, custom or third-party storage connectors or database drivers can be used to expand data storage connectivity.

Another option for engine enhancement is extending SQL (or its dialect) functionality via user defined functions (UDF). The point of UDF is transforming complex composite queries into reusable functions. Once a function is implemented in some programming language, compiled to a library and deployed alongside an engine, it can be called and used as any standard SQL

function. Using UDF makes it easier to operate with complex calculations, nested aggregations and sophisticated requests.

Without doubts, engine customization may be very handy; on the other hand, it requires certain skills to implement them and ensure that it works correctly in all cases. Also, not all engines provide customization capabilities and it should be taken into consideration, especially when it is known from the beginning that needs might require some more specific functionality.

2.3.1.6 Data model required

In order to process the data, an engine needs to know its structure to be able to parse them and get actual information from it. These metadata can be either given to an engine directly or derived by it during reading. In the first case information about data structure and type can be define in a storage layer (for example, HDFS folder structure) or a self-describing file format (such as Parquet). But necessity of explicitly defining and providing this information might be inconvenient when it comes to work with unstructured data or multiple different data storages at the same time. For handling these cases it is practical to use engines implementing i.e. schema on read concept when data are translated into some standartized structure during reading them regardless of its input format.

Thereby, a choice of an SQL engine may depend on a way metadata are handled. If data to process are unstructured or non-uniform and an organization wants to avoid setting up an ETL process to enrich or unify them, then an engine featuring schema on read should be used; although they are usually slower and exacting when it comes to system resources, it still might be more feasible than putting an effort into data preprocessing. On the other hand, metadata may be already provided by the data layout and especially when performance matters, it worth leverage them with an engine optimized to work with it.

2.3.2 Comparison table

The following table sums up an overview of key aspects of all engine studied in this thesis.

	Apache Hive	Apache Impala	Apache Drill	Presto
Deployment model	• on premise • Amazon EMR • Google Cloud Dataproc	• on premise • Amazon EMR	• on premise	• on premise • Amazon EMR
Support vendors	• Cloudera • Hortonworks	• Cloudera	—	• Starburst Data • Teradata
Data sources	HDFS	HBase HDFS	Amazon S3 File System HBase HTTPD Hive Kafka MapR-DB Format MongoDB OpenTSDB RDBMS	Accumulo Amazon Redshift Cassandra Hive Kafka local file system MongoDB RAM RDBMS Redis TPCH/TPCDS
File formats	Avro CSV JSON Parquet	Avro Parquet RCFile SequenceFile Text	Avro CSV/TSV/PSV JSON Parquet SequenceFile	Avro JSON ORC Parquet RCFile SequenceFile Text
System requirements	• Linux/Windows • Java 1.7 • Hadoop	• Linux • MySQL/PostgreSQL • Hive metastore • Java (Oracle JVM)	• Linux/Mac OS X/ Windows • Oracle or OpenJDK 8 • ZooKeeper	• Linux/Mac OS X • Java 8, 64-bit • Python 2.4+
Customization	• UDF • custom file readers	• UDF	• UDF	• UDF • custom data source connectors
Data model	data can automatically translated into Hive tables or uploaded into a predefined schema	data are uploaded into predefined schema	no need to have any predefined schema as schema is discovered on-the-fly	data are uploaded into predefined schema

Figure 2.5: Table comparing chosen engines

2.4 Use case examples

2.4.1 Apache Hive

Presume having various log about all kinds of activity users perform working with an application. It can be timestamps, location codes, items they search for, devices used etc. For a service provider it would be really useful to have resources to discover patterns in users' actions, track down their preferences, determine the most popular items in order to know what exactly is happening inside, how people are using the product, what they are interested in – and Hive can help to do it.

Possible scenario here is to store all log files on HDFS and periodically run analytical jobs either on all existing data or only on separate partitions; Hive is able to consume and process any data which needs to be analyzed regardless of its format. Output of these jobs can be used for creating various reports summing up multidimensional parameters; integrating Hive with business intelligence tools can make this process even easier. As this type of data analytics does not required to have job results immediately, longer execution time does not matter here. Jobs execution can be scheduled and synchronized, for example, with regular data updates when batches of latest records are added either to existing datasets or uploaded to new partitions.

Hive is a powerful tool helping to answer big questions about trends, dynamics, resources usage and other high-level aspects of any activity; it is perfect for batch processing with high throughput and ETL workloads. Therefore, typical use case for it is comprehensive analytics on larger heterogeneous datasets. Hive is capable of aggregating and processing enormous data sets as all queries are converted to separate jobs executed via optimized computing frameworks. Although it makes processing significantly slower, obtained results juxtapose all desired aspects and provides all information which can be extracted from the data.

2.4.2 Apache Impala

Big data analytics technologies unlock new opportunities in the retail sector; probably the most significant of them is predictive analytics on sales data. Findings based on consumers behavior help understand their needs and therefore better meet demand, optimize costs and tailor a product range accordingly. For getting these insights, business users need to have an infrastructure for accessing, exploring and analyzing all sales data which requires robust data analytics platform capable of low-latency processing records from different sources.

When it comes to larger, enterprise-like organizations, Hadoop ecosystem is one of the most popular options for building big data infrastructure. It provides components for data storing, integration, aggregation, task schedul-

ing, service orchestration and resource management. Once data located in Hadoop need to be processed for getting actual insight, Impala can help with interactive exploratory analytics using classic SQL. It has flawless integration with the whole ecosystem and is optimized to share data with other Hadoop components which eliminates latency as there is no need to move data across a cluster. Impala enables harnessing data with running ad hoc queries in real time when results are returned in a matter of minutes or even seconds while multi-user concurrency ensures smooth operating even for bigger number of users. Moreover, full support for Impala is provided by Cloudera, one of the main Apache Hadoop vendors; purchasing this option can help organizations to focus on actual leveraging tools for using data instead of spending time maintaining and troubleshooting it.

2.4.3 Apache Drill

Various Internet of Things (IoT) platforms have been around for several years already implemented in different organizations. In recent times ideas of coupling them with big data have been becoming more and more popular as IoT systems provide a lot of data and metrics naturally and utilizing it with data analytics techniques can unlock new opportunities for leveraging obtained information in a beneficial way.

Consider, for example, data-gathering sensors installed on vehicles used for particular operations. Depending on a vehicle type, they can measure and monitor such parameters as fuel consumption, speed, mileage, idling time, routes taken etc. Gathering and analyzing this information can help to find patterns, correlations, deviations and based on it make predictions about components' state in the near future or find how to optimize the way systems are operating. For example,

- determine when a preventive maintenance should be performed for a particular engine
- find an optimal route to reduce fuel consumption
- rebalance load to avoid some machines being idle while other are overloaded

However, sensor data are often stored in specific formats which also can evolve over time with changing physical appliance or its settings. Besides that, new records are submitted regularly, usually with a high frequency and need to be processed as soon as possible together with those submitted before. Apache Drill suits well to these requirements and therefore can be leveraged effectively in data analytics infrastructures integrated with IoT platforms. It does not require any data preprocessing and is able to query structured and schema-free data. Data structure is discovered right in the moment of reading,

so even input with evolving schema can be processed. Drill is fast and can be used for interactive investigative analytics as well as for operational reporting with leveraging both historical and recently added data at the same time.

2.4.4 Presto

Nowadays decision-making processes are becoming more data driven as an amount of available information to take into account is increasing. Consider, for example, online shopping. Each user opening web pages of online shop is tracked with cookies files storing data about pages he has visited, actions he has performed there, time spent etc. These data are propagated to a database on a backend and can be used for various marketing purposes. For example, if it is not the first time the particular user is browsing through the shop, a special algorithm can propose him similar items he may be interested in or those related to already viewed ones. For choosing them it runs queries against the database to find out what has user been searching before, what he was interested in the most, what are his preferences etc. and based on this information selects items to suggest; it can also search for users with similar preferences and recommend items they have viewed or bought. The same principle works for advertisement shown to a user based on his search queries and pages viewed.

Such tasks require tools for getting quick insights and ad hoc querying capable of running short interactive queries with low latency. In a big data world Presto is a perfect choice for these purposes. It can query various data sources at the same time and provide insights within minutes. Besides its performance, Presto is relatively user-friendly and provides multiple options for managing and using it which makes it accessible not only for data analytics specialists but also for business user.

In general, Presto is good for OLAP type workloads when data need to be analyzed from multiple perspectives and interactive data exploration when users need to answer some specific questions about data quickly. At the same time, it does not require to have all data located at some specific storage which makes it especially useful for running analytics on already existing heterogeneous data.

Conclusion

Constantly growing value of information compels organizations to manage and use data they have to get or keep competitive advantage and adapt to the environment. The only way to transform data into usable insights is to harness them with data analytics which requires leveraging corresponding infrastructure.

Big data concept has become really important since it was a completely new approach bringing number of opportunities, problems to solve and business growth to cover. That therefore led to the emergence of many solutions for implementing big data principles.

Market research has shown that there are dozens of options for building analytical platforms today and it thus might be quite challenging to choose the most suitable components.

The key to establish an effective data processing workflow is to thoroughly define tasks and approaches to their solving along with desired system capabilities. As a central operational element, an SQL engine should be optimized and suitable for given type of workload to provide end users with exact results in an appropriate manner. Naturally, all components have different system requirements which together with requirements to the entire system lead to inevitability of trade-offs. Efficient infrastructure thus requires proper general design to avoid complex orchestration and performance issues.

According to the thesis objectives, several SQL engines have been reviewed and analyzed; based on these findings, a summarizing report comparing them has been created. To define comparison criteria, such aspects of big data analytics systems as design, workflows, patterns and best practices have been studied.

However, it is impossible to predict exact behavior of the entire system without actually using it in production environment. Infrastructure may require partial or complete redesign at any stage of implementation or exploitation as technologies evolve and needs change. Nevertheless, basic principles for choosing technologies remain the same and can be used regardless of or-

ganization type where a resulting system is deployed.

Of course, the analytical platform by itself is not a silver bullet and cannot give answers to all of the questions; it is just a tool giving specialists an ability to inspect data in detail. If being used correctly, then they can make a huge impact on a business.

As big data sphere constantly develops, new software products are being implemented and appear on the market; they implement more sophisticated solutions, are faster and more robust than their predecessors. Approaches and techniques evolve covering more of use cases and solving wider variety of tasks. Analytical systems will be most likely implemented in the future everywhere where there is enough data to process possibly making big data one of IT industry standards.

Every software ever developed was created to solve a specific problem. However, big data opens the door to entirely new discipline of preventing problems and that might be the most significant asset of every single emerging tool at the moment.

Bibliography

[1] Reinsel, D; Gantz, J.; Rydning, J.: Data Age 2025: The Evolution of Data to Life-Critical. 2017. Available from:
https://www.seagate.com/files/www-content/our-story/trends/files/Seagate-WP-DataAge2025-March-2017.pdf
Last accessed on 2017-10-28

[2] Marr, B.: A brief history of big data everyone should read. 2015. Available from:
https://www.weforum.org/agenda/2015/02/a-brief-history-of-big-data-everyone-should-read/
Last accessed on 2017-10-28

[3] CAP theorem – Wikipedia
https://en.wikipedia.org/wiki/CAP_theorem
Last accessed on 2017-11-20

[4] Consistency (database systems) – Wikipedia
https://en.wikipedia.org/wiki/Consistency_(database_systems)
Last accessed on 2017-11-20

[5] Lipcon, T.: Design Patterns for Distributed Non-Relational Databases. Available from: http://cloudera-todd.s3.amazonaws.com/nosql.pdf
Last accessed on 2017-11-28

[6] Brewer, E.: CAP Twelve Years Later: How the "Rules" Have Changed
https://www.infoq.com/articles/cap-twelve-years-later-how-the-rules-have-changed
Last accessed on 2017-11-29

[7] Pritchett, D.: BASE: An Acid Alternative. ACM Queue, Volume 6 Issue 3, 2008: p.48-55

[8] Big data architecture
http://bigdata.teradata.com/US/Big-Ideas/Big-Data-Architecture/
Last accessed on 2018-4-2

[9] George, L.: Getting Started with Big Data Architecture. 2014. Available from http://blog.cloudera.com/blog/2014/09/getting-started-with-big-data-architecture/
Last accessed on 2018-5-9

[10] DIKW pyramid – Wikipedia. Available from: https://en.wikipedia.org/wiki/DIKW_pyramid#/media/File:DIKW_Pyramid.svg
Last accessed on 2017-12-20

[11] Data analysis – Wikipedia
https://en.wikipedia.org/wiki/Data_analysis
Last accessed on 2017-12-5

[12] How companies are using big data and analytics
https://www.mckinsey.com/business-functions/mckinsey-analytics/our-insights/how-companies-are-using-big-data-and-analytics
Last accessed on 2017-12-5

[13] What is data analytics (DA)?
http://searchdatamanagement.techtarget.com/definition/data-analytics
Last accessed on 2017-12-5

[14] What is Data Analytics: Definition – Informatica US
https://www.informatica.com/services-and-training/glossary-of-terms/data-analytics-definition.html
Last accessed on 2017-12-5

[15] Four Types of Big Data Analytics and Examples of Their Use
http://www.ingrammicroadvisor.com/data-center/four-types-of-big-data-analytics-and-examples-of-their-use
Last accessed on 2017-12-5

[16] Moreno, H.: Data Analytics Is No Longer A Nice Option – It's The Core Of The Enterprise. 2017.
https://www.forbes.com/sites/forbesinsights/2017/06/12/data-analytics-is-no-longer-a-nice-option-its-the-core-of-the-enterprise
Last accessed on 2017-12-11

[17] Apache Hive™
https://hive.apache.org/
Last accessed on 2017-12-9

[18] Apache™ Hadoop®official website
http://hadoop.apache.org/
Last accessed on 2017-12-9

[19] Apache Impala official website
https://impala.apache.org
Last accessed on 2017-12-10

[20] Chen, Y. et al.: How Impala Scales for Business Intelligence: New Test Results. 2015.
http://blog.cloudera.com/blog/2015/09/how-impala-scales-for-business-intelligence-new-test-results/
Last accessed on 2018-2-7

[21] Impala Overview
https://www.tutorialspoint.com/impala/impala_overview.htm
Last accessed on 2017-12-10

[22] Kornacker, M. et al.: Impala: A Modern, Open-Source SQL Engine for Hadoop. CIDR 2015. Available from: http://cidrdb.org/cidr2015/Papers/CIDR15_Paper28.pdf
Last accessed on 2017-12-13

[23] Ghat, D.; Rorke, D.; Kumar, D.: New SQL Benchmarks: Apache Impala (incubating) Uniquely Delivers Analytic Database Performance. 2016.
https://blog.cloudera.com/blog/2016/02/new-sql-benchmarks-apache-impala-incubating-2-3-uniquely-delivers-analytic-database-performance/
Last accessed on 2017-12-9

[24] Rahn, G.; Mokhtar, M.: Apache Impala Leads Traditional Analytic Database. 2017.
https://blog.cloudera.com/blog/2017/04/apache-impala-leads-traditional-analytic-database/
Last accessed on 2017-12-9

[25] TPC-DS – Homepage
http://www.tpc.org/tpcds/
Last accessed on 2017-12-9

[26] Setting Up HiveServer2
https://cwiki.apache.org/confluence/display/Hive/Setting+Up+HiveServer2
Last accessed on 2018-1-30

[27] Natkins, J.: How-to: Analyze Twitter Data with Apache Hadoop. 2012.
http://blog.cloudera.com/blog/2012/09/analyzing-twitter-data-with-hadoop/
Last accessed on 2018-1-25

[28] Interface SerDe
https://hive.apache.org/javadocs/r1.2.2/api/org/apache/hadoop/hive/serde2/SerDe.html
Last accessed on 2018-1-25

[29] Hive Language Manual
https://cwiki.apache.org/confluence/display/Hive/LanguageManual
Last accessed on 2018-1-26

[30] Hive Data Definition Language
https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DDL
Last accessed on 2018-1-26

[31] deRoos, D.: Hadoop For Dummies. IDG Books, 2014. ISBN 978-1118607558

[32] Hive Data Manipulation Language
https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DML
Last accessed on 2018-1-26

[33] LanguageManual SubQueries
https://cwiki.apache.org/confluence/display/Hive/LanguageManual+SubQueries
Last accessed on 2018-1-26

[34] Hive Transactions
https://cwiki.apache.org/confluence/display/Hive/Hive+Transactions
Last accessed on 2018-1-26

[35] Apache Hive – Wikipedia. Available from: https://en.wikipedia.org/wiki/File:Hive_architecture.png
Last accessed on 2018-1-27

[36] Hive HPL/SQL
https://cwiki.apache.org/confluence/pages/viewpage.action?pageId=59690156
Last accessed on 2018-1-28

[37] Impala Frequently Asked Questions
https://www.cloudera.com/documentation/enterprise/5-9-x/topics/impala_faq.html
Last accessed on 2018-2-6

[38] Impala Requirements
https://impala.apache.org/docs/build/html/topics/impala_prereqs.html
Last accessed on 2018-2-5

[39] Apache Impala – Overview. Available from https://impala.apache.org/overview.html
Last accessed on 2018-5-9

[40] Impala Schema Objects and Object Name
https://www.cloudera.com/documentation/enterprise/5-4-x/topics/impala_schema_objects.html
Last accessed on 2017-12-10

[41] Using Impala with the Amazon S3 Filesystem https://impala.apache.org/docs/build/html/topics/impala_s3.html
Last accessed on 2017-12-10

[42] Overview of Impala Databases
https://www.cloudera.com/documentation/enterprise/5-4-x/topics/impala_databases.html
Last accessed on 2018-2-6

[43] Overview of Impala Tables
https://www.cloudera.com/documentation/enterprise/5-4-x/topics/impala_tables.html
Last accessed on 2018-2-6

[44] Drill Introduction
https://drill.apache.org/docs/drill-introduction/
Last accessed on 2018-1-10

[45] Architecture – Apache Drill
https://drill.apache.org/architecture/
Last accessed on 2018-1-11

[46] Finley, K.: Google's Real Time Big Data Tool Cloned By Apache Drill. 2012.
https://techcrunch.com/2012/08/17/googles-real-time-big-data-tool-cloned-by-apache-drill/
Last accessed on 2018-1-11

[47] Drill Proposal
https://wiki.apache.org/incubator/DrillProposal
Last accessed on 2018-1-11

[48] Friedman, E.:Apache Drill: Tracking its history as an open source community. 2015.
http://radar.oreilly.com/2015/09/apache-drill-tracking-its-history-as-an-open-source-community.html
Last accessed on 2018-3-22

[49] Architecture Introduction – Apache Drill https://drill.apache.org/docs/architecture-introduction/
Last accessed on 2018-3-22

[50] Apache Zookeeper
https://zookeeper.apache.org/
Last accessed on 2018-2-7

[51] Core Modules – Apache Drill
https://drill.apache.org/docs/core-modules/
Last accessed on 2018-2-7

[52] Connect a Data Source Introduction
https://drill.apache.org/docs/connect-a-data-source-introduction/
Last accessed on 2018-3-19

[53] Frequently Asked Questions – Apache Drill
https://drill.apache.org/faq/
Last accessed on 2018-1-11

[54] Apache Drill – Schema-free SQL Query Engine for Hadoop, NoSQL and Cloud Storage
https://drill.apache.org/
Last accessed on 2018-1-11

[55] Apache Drill
https://mapr.com/products/apache-drill/
Last accessed on 2018-1-11

[56] Presto — Distributed SQL Query Engine for Big Data
https://prestodb.io/
Last accessed on 2017-12-29

[57] Open Source Presto: Now Ready for the Enterprise
http://assets.teradata.com/resourceCenter/downloads/Datasheets/EB8901.pdf
Last accessed on 2018-3-11

[58] Presto (SQL query engine) – Wikipedia. Available from: https://en.wikipedia.org/wiki/Presto_(SQL_query_engine)#/media/File:Wiki_arch.png
Last accessed on 2018-5-9

[59] 1.2. Presto Concepts – Presto 0.200 Documentation
https://prestodb.io/docs/current/overview/concepts.html
Last accessed on 2018-5-9

[60] 4.3. Properties Reference – Presto 0.200 Documentation
https://prestodb.io/docs/current/admin/properties.html
Last accessed on 2017-12-30

[61] 4.4. Spill to Disk – Presto 0.200 Documentation
https://prestodb.io/docs/current/admin/spill.html
Last accessed on 2017-12-29

[62] Business Intelligence Leaders Join with Teradata to Enhance Presto for the Enterprise
https://www.prnewswire.com/news-releases/business-intelligence-leaders-join-with-teradata-to-enhance-presto-for-the-enterprise-300290497.html
Last accessed on 2018-3-18

[63] Presto — Overview
https://prestodb.io/overview.html
Last accessed on 2018-3-4

[64] Amazon EMR – Amazon Web Services
https://aws.amazon.com/emr/
Last accessed on 2018-3-7

[65] 2.2. Command Line Interface – Presto 0.200 Documentation
https://prestodb.io/docs/current/installation/cli.html
Last accessed on 2018-1-2

[66] 4.1. Web Interface – Presto 0.200 Documentation
https://prestodb.io/docs/current/admin/web-interface.html
Last accessed on 2018-1-2

Appendix A

Acronyms

ANSI American National Standards Institute

API Application programming interface

BI Business intelligence

CLI Command-line interface

DML Data modification language

ETL Extract, transform, load

HDFS Hadoop distributed file system

HTTP Hypertext Transfer Protocol

IT Information technology

JDBC Java Database Connectivity

JVM Java virtual machine

ODBC Open Database Connectivity

PL/SQL Procedural Language/Structured Query Language

RDBMS Relational database management system

RPC Remote procedure call

SQL Structed query language

URL Uniform Resource Locator

Appendix B

Contents of enclosed CD

- `readme.txt` ..... the file with CD contents description
- `src` ..... the directory of source codes
  - `thesis` ..... the directory of LaTeX source codes of the thesis
- `text` ..... the thesis text directory
  - `thesis.pdf` ..... the thesis text in PDF format
  - `table.pdf` ..... the comparison table in PDF format