Big Data Ecosystem

Roman Hanzlík

Bachelor's Thesis
2019

Tomas Bata University in Zlín
Faculty of Applied Informatics

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
akademický rok: 2018/2019

ZADÁNÍ BAKALÁŘSKÉ PRÁCE
(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Roman Hanzlík**
Osobní číslo:	**A16663**
Studijní program:	**B3902 Inženýrská informatika**
Studijní obor:	**Softwarové inženýrství**
Forma studia:	**prezenční**
Téma práce:	**Big Data Ecosystem**
Téma anglicky:	**Big Data Ecosystem**

Zásady pro vypracování:

1. **Zpracujte literární rešerši na dané téma.**
2. **Proveďte popis jednotlivých vybraných technologií.**
3. **Vypracujte pro ně praktické návody k instalaci a konfiguraci.**
4. **Vytvořte funkční sady demonstračních příkladů pro nejběžnější použití daných technologií.**
5. **Proveďte celkové zhodnocení a závěr.**

Rozsah bakalářské práce:	
Rozsah příloh:	
Forma zpracování bakalářské práce:	**tištěná/elektronická**

Seznam odborné literatury:

1. **ERL, Thomas, Wajid KHATTAK a Paul BUHLER. Big data fundamentals: concepts, drivers & techniques. Vancouver, BC: Service Tech Press, [2016]. ISBN 978-0-13-429107-9.**
2. **WALKER, Russell. From big data to big profits: success with data and analytics. New York, NY: Oxford University Press, [2015]. ISBN 978-0199378326.**
3. **HARRISON, Guy. Next generation databases: NoSQL, NewSQL, and Big Data. New York, NY: Apress, [2015]. ISBN 978-148421330.**
4. **SITTO, Kevin. Field guide to Hadoop. USA: O'Reilly, [2015]. ISBN 978-1491947937.**
5. **VOHRA, Deepak. Practical Hadoop ecosystem: a definitive guide to Hadoop-related frameworks and tools. New York, NY: Apress, [2016]. ISBN 978-1484221983.**

Vedoucí bakalářské práce:	**doc. Ing. Roman Šenkeřík, Ph.D.** Ústav informatiky a umělé inteligence
Datum zadání bakalářské práce:	**3. prosince 2018**
Termín odevzdání bakalářské práce:	**15. května 2019**

Ve Zlíně dne 7. prosince 2018

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta aplikované informatiky
①
L.S.

doc. Mgr. Milan Adámek, Ph.D.
děkan

prof. Mgr. Roman Jašek, Ph.D.
ředitel ústavu

ABSTRAKT

Cílem této bakalářské práce je představení technologií Big Data a jejich možné využití v praxi. Hlavním cílem této práce v teoretické části je poskytnout přehled v oblasti Big Data a NoSQL databází pro všechny začátečníky. V praktické části je detailně představen Apache Hadoop a projekty s ním svázané. Součástí práce jsou postupy, jak dané technologie nainstalovat včetně příkladů a scénářů použití.

Klíčová slova: Big Data, NoSQL, CAP teorém, Apache Hadoop, HDFS, YARN, MapReduce, Apache Hadoop Ecosystem, Spark, Pig, Hive, Sqoop

ABSTRACT

This bachelor‘s thesis presents Big Data technologies and their possible real-world use cases and applications. The main goal of this thesis is to provide the first introduction to Big Data and NoSQL databases for newcomers. In practical part, Apache Hadoop and its surrounding projects are presented in detail. Integral part of this thesis is light cookbook how to install particular technologies itself with functional demo examples of possible use cases and scenarios.

Keywords: Big Data, NoSQL, CAP Theorem, Apache Hadoop, HDFS, YARN, MapReduce, Apache Hadoop Ecosystem, Spark, Pig, Hive, Sqoop

MOTTO

Study the past if you would define the future.
Confucius

Scientia potentia est, sapientiae privilegium.
[Knowledge is power, wisdom privilege.]
Sir Francis Bacon/Roman Hanzlík

Where is the wisdom we have lost in knowledge?
Where is the knowledge we have lost in information?
Thomas Stearns Eliot

CONTENTS

INTRODUCTION

Look around, and tell me what do you see? One could say “Cities, fields, mountains ... “, other one “Twittering of the birds, the noise of the rivers, a hastiness of people ...” and I? I see the data all around. Data in form of temperature, wind, humidity when I wake up and open the window. I see the stock market price in the morning news, when I switch the TV on. I see the traffic data during my commute to the office. I see the data behind the attendance system, once I check in. I see the data in surveillance cameras. I see the data in Twitter and Facebook, every time I connect. I see … I see the data everywhere. Working with data for almost 25 years I just DO see them.

When I was a kid, I remember myself doing tables of ski races results and collecting them, in that time just collecting. When I became older, I did tables of soccer matches scores of most of the national leagues and try to predict next matches results. And when I left the high school and went to business I surrounded myself with data or better said, data just surrounded me. It started as a hobby and turned into passion, passion for data. Even now, when I look at you, I see data, I can utilize. Data are like time, they’re with us from the very beginning. Data flow with time like inseparable twins, they’re getting older, one faster than the other. One piece of data is old and useless the very next second, the other one could be valuable for a long time even forever.

There were 4 milestones in the era of human which dramatically change the amount of data recording. First - c. 6000 BC start of using language and writing, second - Gutenberg’s printing press in mid of 15th century, third - 40s of 20th century invention of computers; and finally, fourth – Internet and surrounding technologies. At each of these evolution steps, the amount of data increased dramatically. The first record of data being reliably stored in a mechanized medium was using paper tape in 1846 by Alexander Bain – the inventor of the electric printing telegraph [1]. Since then, with every single year the technology innovations continue to accelerate the data in diverse characteristics. Not only volume, but also velocity and variety of data increased in last couple of years exponentially, traditional approaches are not able to fulfil expectation of businesses or governments, everything is getting bigger and faster, so the new technologies must come on board. We are going to discuss the volume of the data from historical perspective deeply in further chapter.

Structure of Thesis

The structure and outline of this thesis contains two parts. In the first part, the theoretical aspects of Big Data technologies and NoSQL databases are presented with their possible real-world use cases and applications. In the second, practical part, the Apache Hadoop Ecosystem is presented and explained in detail, with practical installation and configuration cookbook and demo examples how to use them.

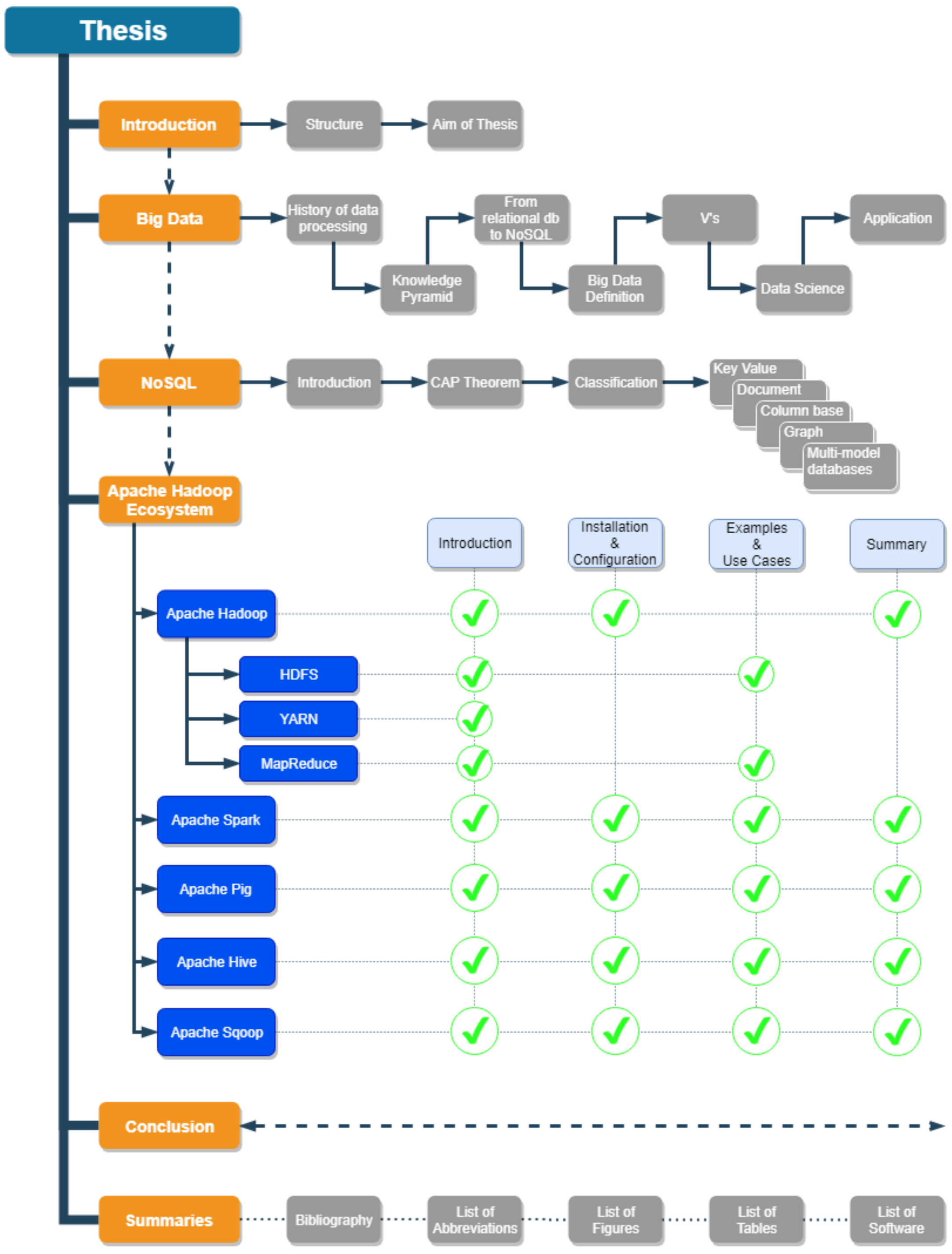


Figure 1 – The structure of the Bachelor's thesis

Theoretical part initially gives an overview of the data history, information processing and presents factors which led to need for this new approach, continues with the definition of Big Data and the V's of Big Data. Next, in the second section of theoretical part, NoSQL are presented together with CAP theorem, and the most popular and used NoSQL databases from each type such as Key-value stores, Wide-column stores, Document databases, and Graph databases are listed.

The practical part starts with description of the big picture of Apache Hadoop Ecosystem, continues with detail presentation of Apache Hadoop itself. Main principles of Hadoop are described, such as a distributed storage (HDFS), a resource management (YARN) and a data processing (MapReduce). Step-by-step installation and configuration instructions are also provided as well as for other chosen technologies from surrounding ecosystem. For each particular technology exists a set of demo examples with the functional lines of codes to demonstrate usage of technology.

Finally, in the end, the conclusion of the thesis summarizes this work. Bibliography, list of figures and tables and list of used software and their versions are integral part of this work.

Aim of Thesis

The primary aim of this thesis is to extend my current knowledge of area around Datawarehouse, Business Intelligence and Datamining to new emerging technologies of Big Data so I could in my consulting practice provide broader knowledge and deeper expertise to my customers.

The next one, equally important goal of this thesis is to provide the first introduction to Big Data and NoSQL databases to all students at Tomas Bata University in the new course started 2019/2020 named "Advanced Database Systems" with light cookbook how to install particular technologies itself with examples of possible use cases and scenarios.

Last but definitely not least goal is to build the technology background, so I can continue on top of it in Master and/or Doctoral study in the area of Data Science.

I. THEORY

1 BIG DATA

1.1 History of data processing

From the very beginning (2.8 mil years ago) the *Homo* {„human being“} genus differs from all other species around. Homo evolved through *Homo habilis* (2.3 – 1.4 mil years ago), *Homo erectus* (1.8 mil – 30,000 years ago), *Homo neanderthalensis* (400,000 – 40,000 years ago) into a current version of us: *Homo sapiens* specifically *Homo sapiens sapiens* {„wise man“} (280,000 years ago until now). What differs this genus from others? Most of the species transfer their knowledge through learning descendant from their parents. Human is the one who transfers his knowledge via teaching and is a part of the other species who also teach their descendant differ with precision of the transfer. The next thing, and definitely the one what gives to human biggest advantage was a gradual ability to speak. It's estimated that around 100,000 years ago Homo started using language - first basic interjection of pain or happiness, later nouns of things in surrounding nature and last with verb of daily activities. This research [2] indicates that some Homo species had the ability to produce speech sounds that overlap with the range of speech sounds of modern humans, and that species such as Neanderthals possessed genes that play a role in language of humans. Next big milestone - human started to record its information in form of pictures. The oldest pictures in caves are dated about 30,000 years ago. Next logical step (for us in these days, not in that time) was evolution of pictures into symbols what can be called *first writing system* used mostly for recording and later also for communication. Writing preceded first with pictogram and ideograms. The best-known examples are Jiahu symbols carved on tortoise shells in Jiahu (6600 BC), Vinča symbols sometimes called Danube script (Tărtăria tablets, c. 5300 BC). The invention of the first writing systems is roughly dated as of the late 4th millennium BC. The Sumerian cuneiform script and the Egyptian hieroglyphs are generally considered the earliest writing systems (c. 3400 to 3200 BC). It is generally agreed that Sumerian writing was an independent invention; however, it is debated whether Egyptian writing was developed completely independently of Sumerian or was a case of cultural diffusion. The similar debate exists for Chinese script developed around (1200 BC). So-called modern language appeared in the archaeological record only recently, with the advent of lexicographic writing around 5,400 years ago [3].

The oldest record of counting is dated from around 5000 BC, when Sumerian merchants used small clay beads to denote goods for trade. Counting on a larger scale, was the privilege of the state, when governments for millennia have tried to keep track of their people by collecting information. The ancient Egyptians conducted census, as did also the Chinese. They're mentioned in the Old Testament, and the New Testament tells us that a census imposed by Caesar Augustus — "that all the world should be taxed" (Luke 2:1) — took Joseph and Mary to Bethlehem, where Jesus was born [4].

With both these abilities (speaking and writing same language) started the Age of Data. Ability to express own ideas led cultures to collect information and share knowledge and wisdom to next generations.

1.2 Knowledge Pyramid

To understand Big Data, it's necessary to start with defining some terms like data, information, and knowledge, describe their context and continuity.

DIKW Pyramid, Knowledge Pyramid, Wisdom Hierarchy, Information Hierarchy are some of the names referring to the popular representation of the relationships between data, information, knowledge and wisdom. Each step up the pyramid graphically represented in Figure 2 answers the questions about the initial data and adds value to it. The more questions we answer, the higher we move up the pyramid. In other words, the more we enrich our data with meaning and context, the more knowledge and insights we get out of it. At the top of the pyramid, we have turned the knowledge and insights into a learning experience that guides our actions.

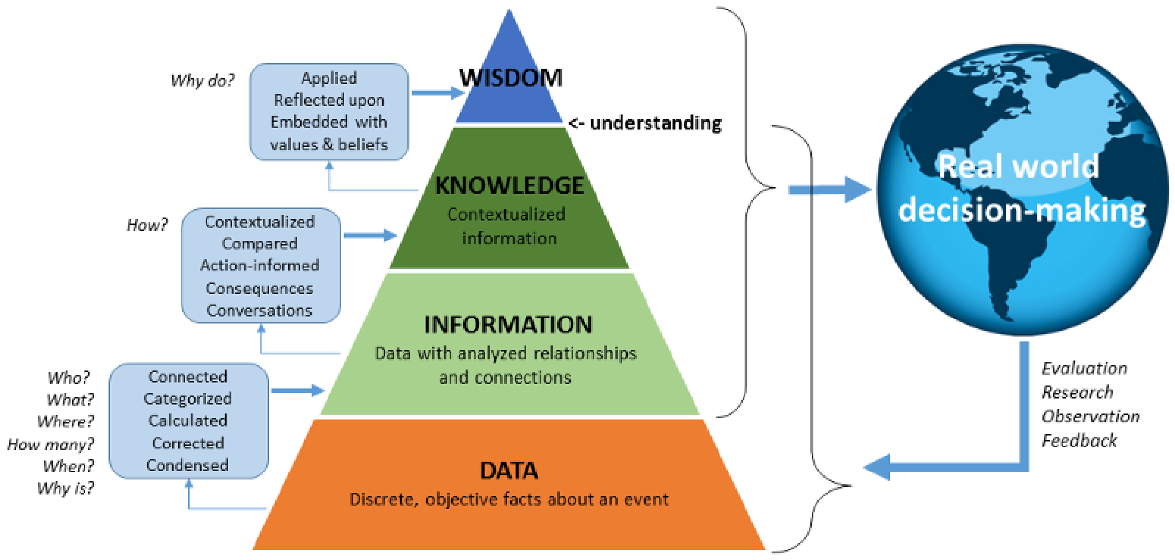


Figure 2 – Knowledge Pyramid [5]

Data – a collection of facts in a raw or unorganized form. In informatics, text, number, picture, sound, or everything we can process automatically with machines are understood as data. However, mostly without any context, data can mean little.

Data could be divided into different groups by several conditions like their structure, origin or what they are intended for. In the Table 1 is shown how data are classified by their structure.

Table 1 – Data by their structure

Group	Description	Example
Structured	data in predefined structure	Relational data
Semi-structured	not predefined structure, can be process them easily	XML
Unstructured	data with undefined internal structure	PDF

Structured data – are data whose elements are addressable for effective analysis or further processing. The data are organized into a formatted repository (typically a database). It concern all data which can be stored in relational database in the form of table with rows and columns. They have relational key and can be easily mapped into pre-designed fields. Today, these kind of data are most processed and easily managed information. Structured data represent only 5 to 10% of all informatics data. Examples of these kind of data could be: Meta-data (Time and date of creation, File size, Author etc.), Library Catalogues (date, author, place, subject etc.), Census records (birth, income, employment, place etc.), Economic data (rates, GDP, VAT etc.).

Semi-structured data – are data which do not reside in a relational database but that have some organizational properties that make it easier to analyze. With some process, you can store them in the relational database, on the other hand it could be very hard for some semi-structured data. Semi-structured data also represent only 5 to 10% of all informatics data. Examples of these data could be e.g. personal data stored in XML or JSON files.

Unstructured data – are data that are not organized in a pre-defined manner or does not have a pre-defined data model, thus they are not a good fit for a mainstream relational database. This is one of the reason, why there are alternative platforms for unstructured data. These data increasingly prevail in IT systems and are used by organizations in a variety of business intelligence and analytics applications. Today, more than 80-95% of all data that exist globally are estimated to be unstructured data. Examples of these data could be e.g. user-generated content from social media (e.g., Facebook, Twitter, Instagram, and Tumblr), images, videos, surveillance data, sensor data, call center information, geolocation data, weather data, economic data, government data and reports, research, Internet search trends, and web log files [6] [7].

Another possibility how could be data distinguish is by their origin, so they're divided into External/Internal, Primary/Secondary as listed in the Table 2.

Table 2 – Data by their origin

Group	Description	Example
External	Data from public sector, or external companies	Government data
Internal	Data originated internally	Orders, transactions
Primary	Data originated in the system	Customers in CRM
Secondary	Data taken from other systems	ERP sales reports

Information – is the next building block of the DIKW Pyramid. These are data that have been "cleaned" of errors and further processed in a way that makes it easier to measure, visualize and analyze for a specific purpose. Depending on this purpose, data processing can involve different operations such as combining different sets of data (aggregation), ensuring that the collected data are relevant and accurate (validation), etc. By asking relevant questions about 'who', 'what', 'when', 'where' etc., can be derived valuable information from the data that could be more valuable for us [8].

Knowledge – when is placed the question of 'how', this is what makes the leap from information to knowledge. "How" are the pieces of information connected to other pieces to add more meaning and value to us is the 'Knowledge'.

Wisdom – is knowledge applied in action. We can also say that, if data and information are like a look back to the past, knowledge and wisdom are associated with what we do now and what we want to achieve in the future [8].

1.3 From relational databases to NoSQL

To explain motivation for emergence of NoSQL databases, it's necessary to go back and look at the traditional approach of building applications with the current mainstream usage of the persistent storage. For the past 30 years, applications have been backed using an RDBMS (Relational Database Management System) with a predefined schema that forces data to conform to a schema on-write. Many people still think that they must use an RDBMS for applications, even though records in their data sets have no relation to one another. Additionally, those databases are optimized for transactional use, and data must be exported for analytics purposes. NoSQL technologies have turned that model on their side to deliver groundbreaking performance improvements [9].

The key impulse for emergence wider using alternative DBMS (Database Management System) is dynamic growth of development of global business opportunities and business models based on providing services and selling goods to millions of customers all around the world. In the last few years the source of data become data captured from devices like IoT (Internet of Things). With increasing data volumes and numbers of users, it is becoming more difficult and costly to respond to this situation by scaling vertically, by investing in more and more capacity of database servers. Therefore, there is a need to address this problem by scaling horizontally, by running hundreds to thousands of commodity servers, which at the same time must ensure the continued availability of services with adequate data consistency, consistent with the specific strategy, objectives, content and business processes of the enterprise. Google and Amazon were among the first companies to meet these limits due to their activities. Both companies have created their own scalable database systems based on searching data by keys. Bigtable (Google) [10] and Dynamo (Amazon) [11], respectively database systems described in these papers, are considered the key impulses that inspired and influenced the emergence and further development of NoSQL DBMS. Google's Bigtable database system is a scalable distributed structured data repository run on several thousand commodity servers. Google uses Bigtable for a variety of applications (such as web page indexing, Google Earth) with varying data storage requirements. Stored data range from individual URLs of web content to satellite imagery of Earth.

The second of the above-mentioned companies, which came up with their own persistent data storage solution, based on principles different from relational DBMS and allowing horizontal scalability, was Amazon. In the second half of the first decade of this century, Amazon was in a position to run one of the largest e-commerce platforms with ten thousands of servers in several data centers around the world. Such infrastructure was necessarily associated with server downtime or, in more severe cases, whole data centers downtime. Even in the event of disruptions to individual servers or part of the infrastructure, it was necessary to ensure the availability of services, as even the slightest outage could mean serious financial implications and permanently shake customers' trust. For example, it was necessary to ensure that customers still see items placed in the shopping cart in the event of any downtime. Amazon addressed these requirements by creating the Dynamo database system, which ensures continued availability of core services. The system worked on the key-value principle, extensively uses versioning of the objects. Unlike Bigtable, which allows applications to query multiple-attribute data, Dynamo was designed for applications that need simple data access based on a single key and with primary focus on a high degree of availability, where interruption of connection with a part of the network infrastructure or server failure does not reject UPDATE operations [11].

Soon other non-relational database systems began to emerge, generally named "NoSQL database". Many of them were inspired by the above mentioned Bigtable and Dynamo projects. For example, one of the systems is Cassandra, database developed by Facebook. NoSQL is not associated with any fixed definition, but the history of its origin can be mapped, and its current understanding is defined quite precisely. For the first time, the notion of NoSQL was used in the late 1990s, paradoxically as the name of a relational database for the UNIX operating system, which dates back to 1998 [12]. However, this act has nothing to do with NoSQL databases in the current concept. The new meaning of NoSQL is attributed to the software developer Johan Oskarsson, who held a meeting in 2009 to explore the latest trends in new non-relational database technologies inspired by the already mentioned Bigtable and Dynamo. Oskarsson tried to find a concise and easy-to-remember name for the upcoming event. From the suggestions sent through the #cassandra IRC channel, he chose the name "NoSQL", which quickly took over, not just as the name of the meeting, but even unintentionally as an aggregate term referring to a whole set of database systems. The vast majority of the professional public agree that the first two letters of the "NoSQL" acronym do not mean "no SQL" but a "not only SQL" definition. While there is no official definition of this term, it can be said that it is a group of databases with certain common attributes: a non-relational model, very suitable for open-source clustering, without a fixed data schema [13].

1.4 Motivations to use NoSQL databases

NoSQL's *shared nothing architecture* (distributed computing architecture in which each node is independent and self-sufficient (none of the nodes share memory or disk storage), and there is no single point of contention across the system) makes it easy to support scale-out architecture, whether on-premises or in the cloud, to deliver high linear scalability. NoSQL is suitable for a variety of scale-out applications including social network, customer analytical workloads, real-time reporting, embedded database applications, mobile applications, the internet of things (IoT), and gaming applications.

NoSQL supports flexible data models to accommodate any type of data. Unlike relational databases, NoSQL can accommodate structured, semi-structured, and unstructured data to support new types of business applications. NoSQL offers more flexible approach in which the application, rather than the datastore, defines the schema and access paths. NoSQL supports a wide range of new data types, including textual types such as JSON as well as many other unstructured and semi-structured data types.

NoSQL delivers an extreme read and write capabilities for demanding customer apps. Applications that demand extensive data reads and writes, often experience excessive latency input/output (I/O) bottlenecks, especially when the app has high volumes of both. NoSQL database management systems are very efficient in reading very large amounts of data.

NoSQL simplifies data management for any type of application. Not only deploying traditional databases takes time and effort – managing them requires specialized database administrators (DBAs) and designing data structures can also take months. NoSQL accelerates deployment through automation and simplification processes, provisioning capabilities, and flexible data structures. When using NoSQL, application developers have complete control over data storage and access and don't need a DBA to support the NoSQL data store.

NoSQL lowers data management cost. Many NoSQL solutions are open source, and others sell for much less than a full version of a commercial relational DBMS. Compared with conventional DBMSs, enterprises report that NoSQL products saved them more than 50% of the cost [14].

1.5 Definition of Big Data

Back in 2013 Dan Ariely's likened Big Data to "*Big data is like teenage sex: everyone talks about it, nobody really knows how to do it, everyone thinks everyone else is doing it, so everyone claims they are doing it.*" From that time many people tried to define or box this new phrase, but the market is so fast and changing that we can't still express exact definition. I see Big Data as area within Information Technology industry which took place to analyze, systematically extract information from, or otherwise deal with data sets that are too large or too complex to be dealt with by traditional data processing application software within the meaning of volume or/and time. In the early 2010s there was also a big discussion regarding if Big Data is next big trend, if companies should take care of it. Now almost 10 years later I would modify Dan Ariely's quote from 2013 to something like "*Big Data is like university student sex: Everybody talks about it, most of them claims they are doing it regularly, only some of them really know how to do it. No one says he never heard about it. Almost everybody looking for other who has know-how and real experience with it.*"

1.6 How big is Big Data

To have a rough estimate how big are Big Data, first look into the Table 3 for some examples of data volume related to metric prefixes.

Table 3 – Example of data volume

Unit	Value		Example
kilo	10^3	1 000	a paragraph of a text document
mega	10^6	1 000 000	a small novel
giga	10^9	1 000 000 000	Beethoven's 5th Symphony
tera	10^{12}	1 000 000 000 000	all the X-rays in a large hospital
peta	10^{15}	1 000 000 000 000 000	half of all US academic research libraries
exa	10^{18}	1 000 000 000 000 000 000	20% of the words ever spoken by human
zetta	10^{21}	1 000 000 000 000 000 000 000	grains of sand on all the world's beaches

Most of us can probably imagine size of mega or giga, maybe some of us even worked with data of terabytes in volume, but petabytes is for most people like 4th dimension in Euclidean geometry. To have a better idea, it's necessary to look at the volume of data from different perspective or to cut one of the dimensions – time – to smaller scale. As you can see in Figure 3Figure 3 – How much data is generated every minute and Figure 4Figure 4 – What happens online in 60 seconds the amount of data generated every minute is enormous.

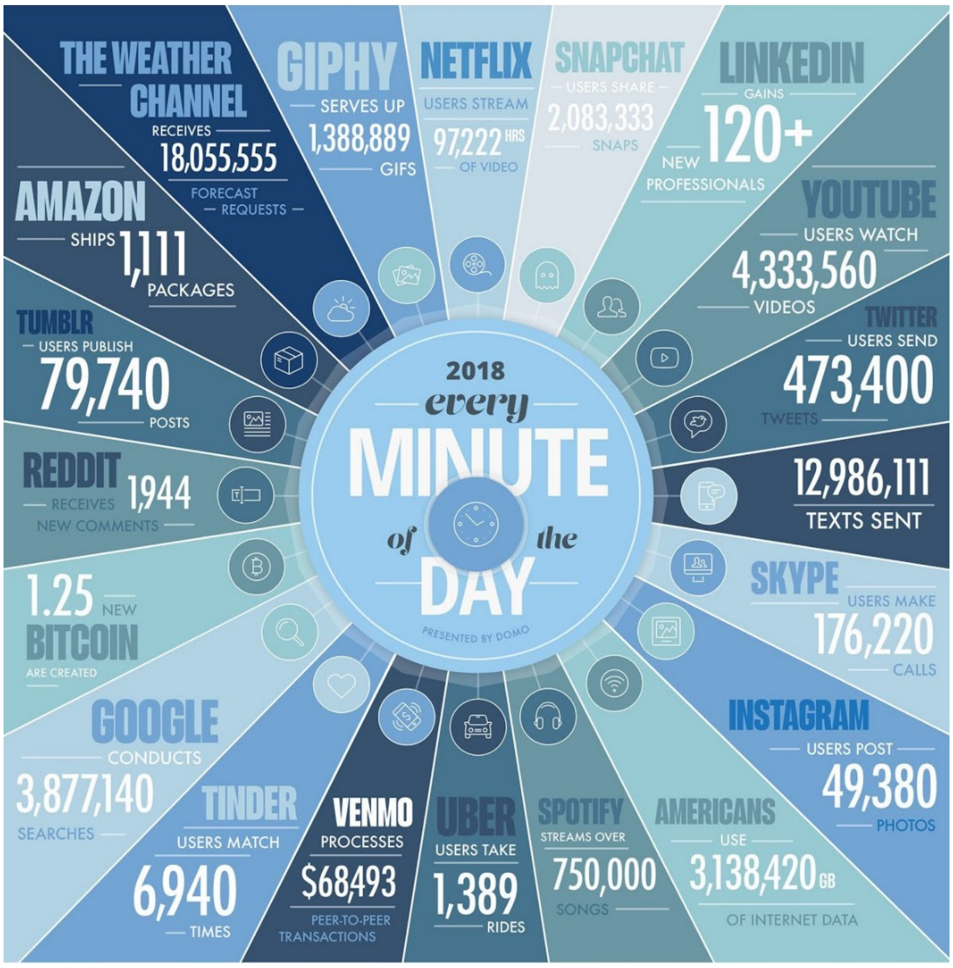


Figure 3 – How much data is generated every minute [15]

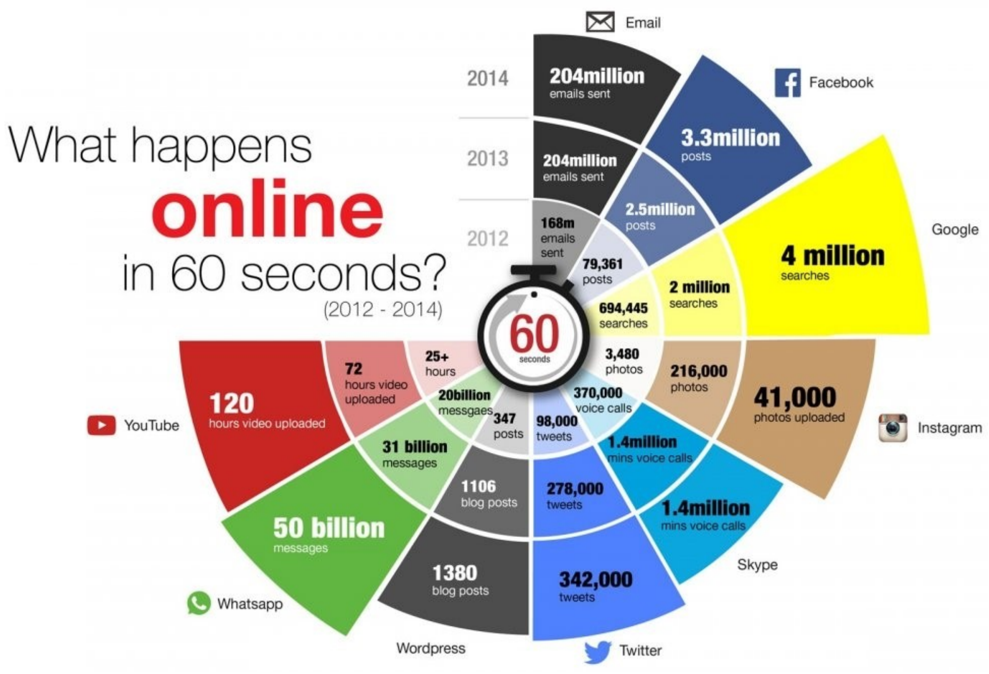


Figure 4 – What happens online in 60 seconds [15]

The Figure 5 depicts the relative increase of data stored over time, together with the diversity and number of users creating and using data, and the variety of use cases for data. Complexity and cost are two other important factors that continue to increase. Therefore the disciplines of Big Data are taking the key role in the 21st century.

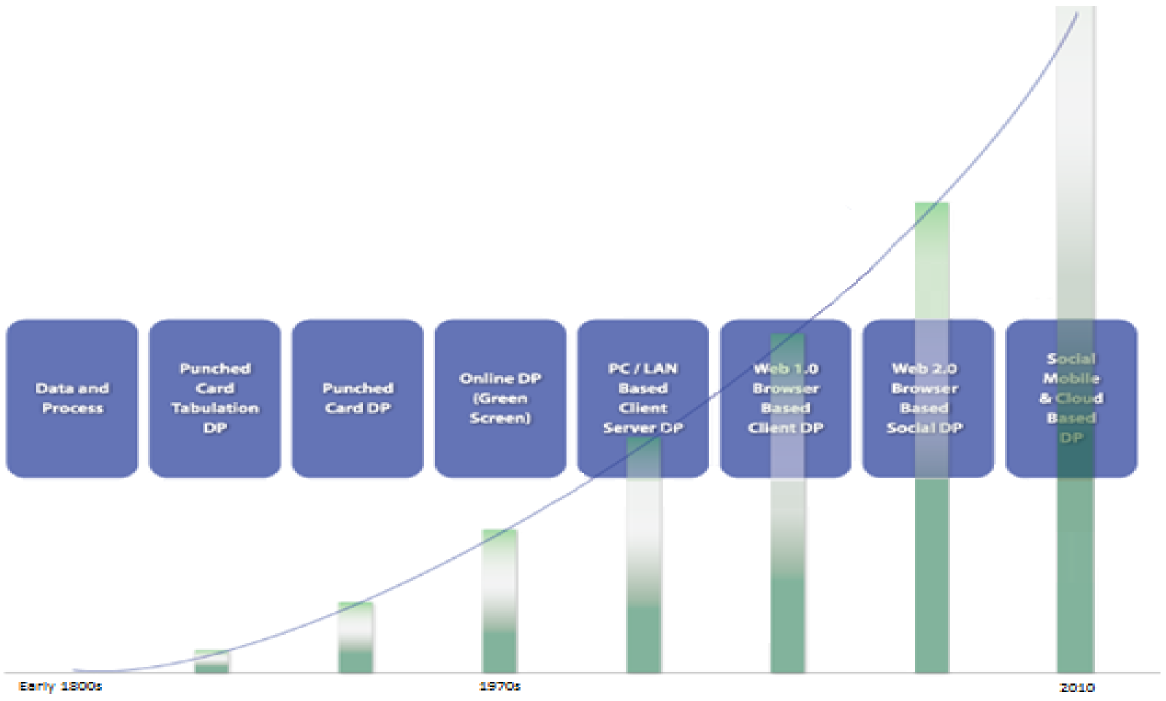


Figure 5 – Data Volume (1800s – 2010) [17]

To have an idea about the volume increase in long-term time horizon, see Figure 6, where you can see data volume in last 10 years with prediction for next 6. Any further prediction especially in IT sector would be just divination from a crystal ball.

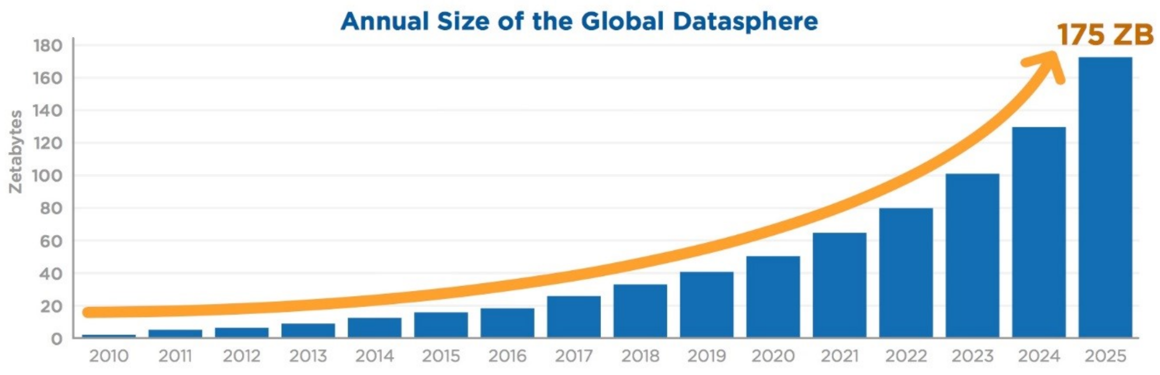


Figure 6 – Data Volume (2010 – 2025) [18]

Definitely we can agree that imagine the volume of data in peta or zetta bytes is like imagine how far nearest Black hole is. To help you with this, following example of the amount of data generated worldwide this year should definitely help you.

Source data:

- 40 zettabytes of data generated this year = 40,000,000,000,000,000,000,000 bytes
- stack of 100 of 3.5” floppy disks measures 13 inches (33 cm) ⇒ 0.33 cm / 1 floppy disk
- capacity of one 3.5” floppy disk is 1.44 MB = 1,474,560 bytes

Calculation:

- # floppy disks to store all data generated this year

40,000,000,000,000,000,000,000 bytes of data / 1,474,560 capacity of 3.55” floppy disk = 27,126,736,111,111,111

- stack height of all floppy disks with data

27,126,736,111,111,111 floppy disks * 0.33 cm = 8,951,822,916,666,666 cm = 89,518,229,166 km

- # of stacks between Earth and Sun

89,518,229,166 km / 14,960,000 km (distance between Earth and Sun) = **600**

To sum it up, all data generated this year worldwide could be stored in 600 stacks (towers) of 3.5” floppy disk from Earth to Sun.

To complete the question „How big are Big Data“, it’s necessary to mention the key factor for storing the data and it’s the cost of storage. Figure 7 shows cost of 1 gigabyte of storage from 1980 till 2014. For the past 35+ years or so, hard drives prices have dropped, from around $500,000 per gigabyte in 1981 to less than $0.03 per gigabyte today. In the last 5 years (2014-2019) the price dropdown is slowing and stabilized between $0.04 - $0.03 [19].

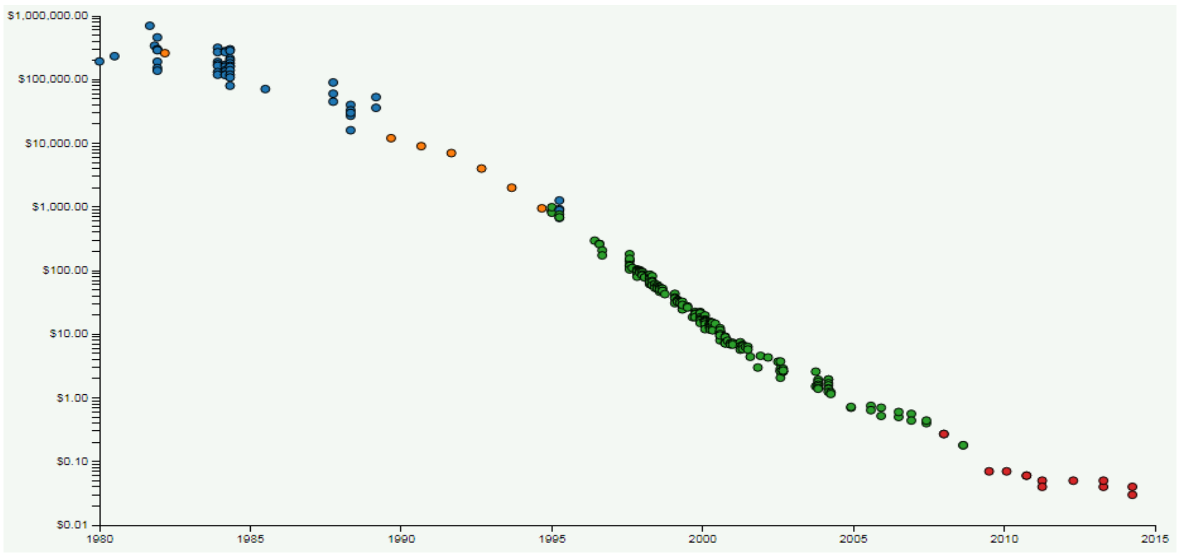


Figure 7 – History of storage cost per gigabyte (1980-2014) [20]

1.7 The V's of Big Data

In the context of large amounts of data, it is often referred to as so-called 3 V's, which are generally recognized as characteristics of Big Data (for graphical representation of these characteristics please see Figure 8). These distinctive 3 V's were defined in 2001 by Dough Laney, analyst in Gartner [21].

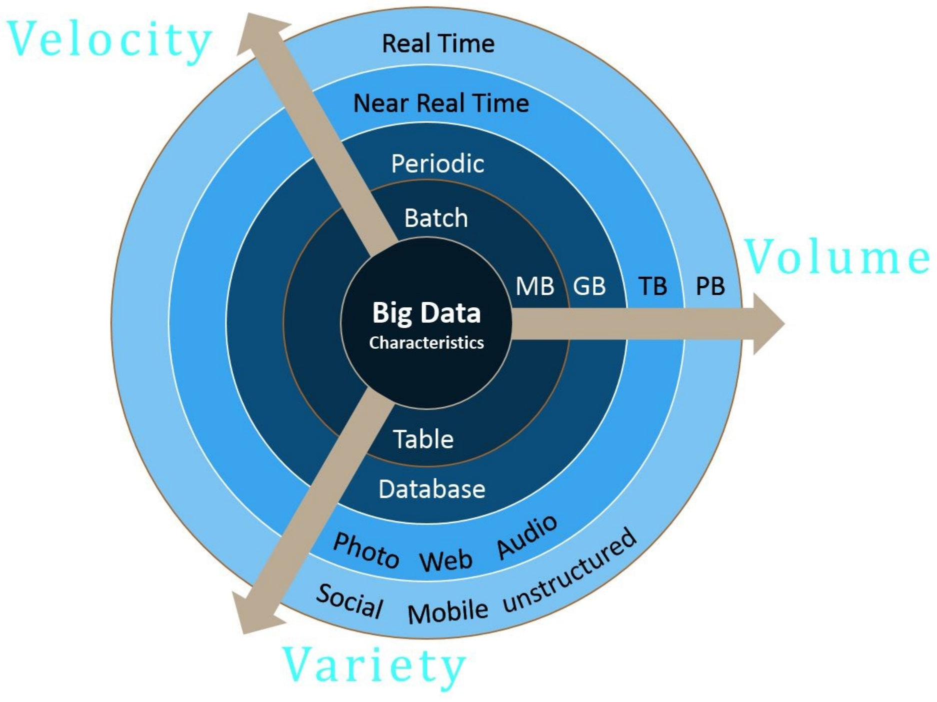


Figure 8 – The 3 V's of Big Data [22]

Volume - Data Size

The first "V" is the *volume*, i.e. the size of the data being processed. With the ever-increasing amount of data generated by modern technologies, the question of how to process these data so that the analysis of these data results in the most inaccurate results in the shortest possible time arose. Such data volume cannot be processed by traditional database tools. What size of data already means "Big" is not said anywhere, but this boundary is constantly shifting as technology evolves.

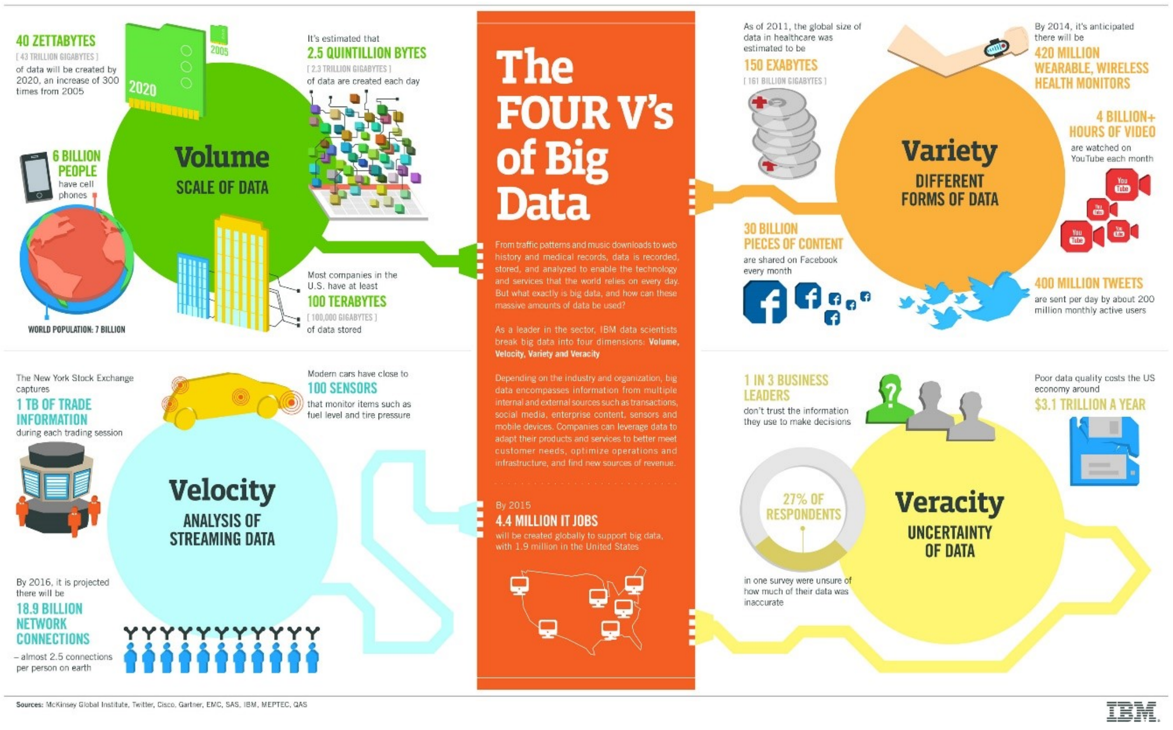
Velocity - Data Rate

The second "V" is the speed at which the data are produced. As data size increases, the speed of data that an organization produces, processes, or just receives increases. The ability to quickly receive and respond to data can be a significant competitive advantage or even a necessity. However, in order for an organization to respond quickly to data, fast data processing is necessary. For traditional database tools, it is very difficult to process unstructured data or structured data of large volumes, the cost of such processing and storage would be unbearable. In contrast, Big Data technologies allow this storage and processing and are designed for it.

Variety - Data Diversity

The third "V" represents the diversity of data. Data rarely come from a single source and are often not in a unified format. This diversity of data is a major difference from traditional data storage methods, where fixed-structure data are used - structured data. Structured data are described by their metadata, and analysis can be easily performed by using traditional database systems. Opposite examples are unstructured data that are neither described nor organized, so it is not easy to process. This type of data occurs, for example, in emails, documents, or social networks. However, in the case of Big Data, it is also possible to analyze such data from different sources and with different structures, which can be a great benefit to organizations.

To these basic 3 V's characteristics of Big Data some authors added later another V's so-called 4 V's / 5 V's as is visualized in Figure 9 and Figure 10.


40 ZETTABYTES
[43 TRILLION GIGABYTES]
of data will be created by 2020, an increase of 300 times from 2005
2020
2005
6 BILLION PEOPLE
have cell phones
WORLD POPULATION: 7 BILLION
Volume
SCALE OF DATA
It's estimated that
2.5 QUINTILLION BYTES
[2.3 TRILLION GIGABYTES]
of data are created each day
Most companies in the U.S. have at least
100 TERABYTES
[100,000 GIGABYTES]
of data stored
The New York Stock Exchange captures
1 TB OF TRADE INFORMATION
during each trading session
Modern cars have close to
100 SENSORS
that monitor items such as fuel level and tire pressure
Velocity
ANALYSIS OF STREAMING DATA
By 2016, it is projected there will be
18.9 BILLION NETWORK CONNECTIONS
– almost 2.5 connections per person on earth
The FOUR V's of Big Data
By 2015
4.4 MILLION IT JOBS
will be created globally to support big data, with 1.9 million in the United States
As of 2011, the global size of data in healthcare was estimated to be
150 EXABYTES
[161 BILLION GIGABYTES]
30 BILLION PIECES OF CONTENT
are shared on Facebook every month
Variety
DIFFERENT FORMS OF DATA
By 2014, it's anticipated there will be
420 MILLION WEARABLE, WIRELESS HEALTH MONITORS
4 BILLION+ HOURS OF VIDEO
are watched on YouTube each month
400 MILLION TWEETS
are sent per day by about 200 million monthly active users
1 IN 3 BUSINESS LEADERS
don't trust the information they use to make decisions
27% OF RESPONDENTS
in one survey were unsure of how much of their data was inaccurate
Veracity
UNCERTAINTY OF DATA
Poor data quality costs the US economy around
$3.1 TRILLION A YEAR
IBM

From traffic patterns and music downloads to web history and medical records, data is recorded, stored, and analyzed to enable the technology and services that the world relies on every day. But what exactly is big data, and how can these massive amounts of data be used?

As a leader in the sector, IBM data scientists break big data into four dimensions: **Volume, Velocity, Variety and Veracity**

Depending on the industry and organization, big data encompasses information from multiple internal and external sources such as transactions, social media, enterprise content, sensors and mobile devices. Companies can leverage data to adapt their products and services to better meet customer needs, optimize operations and infrastructure, and find new sources of revenue.

Sources: McKinsey Global Institute, Twitter, Cisco, Gartner, EMC, SAS, IBM, MEPTEC, QAS

Figure 9 – The 4 V's of Big Data [23]

Veracity – Data Credibility

Credibility is highlighted for possible poor data quality at source. This non-quality and unreliability of the data is then transferred to the analysis, which can result in distorted or non-full results.

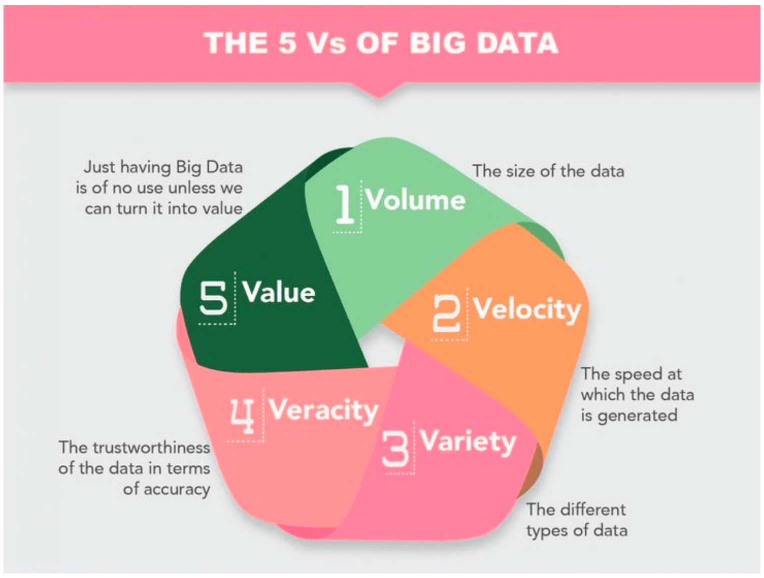

THE 5 Vs OF BIG DATA
1 Volume
The size of the data
2 Velocity
The speed at which the data is generated
3 Variety
The different types of data
4 Veracity
The trustworthiness of the data in terms of accuracy
5 Value
Just having Big Data is of no use unless we can turn it into value

Figure 10 – The 5 V's of Big Data [24]

Value – Data Usefulness

The term value is often understood as the most important one, because the result of the analysis should bring added value and be useful for making the business more effective.

Very soon there was race who will discover next V's so the competition continued with other V's, such as 8 V's (see Figure 11).

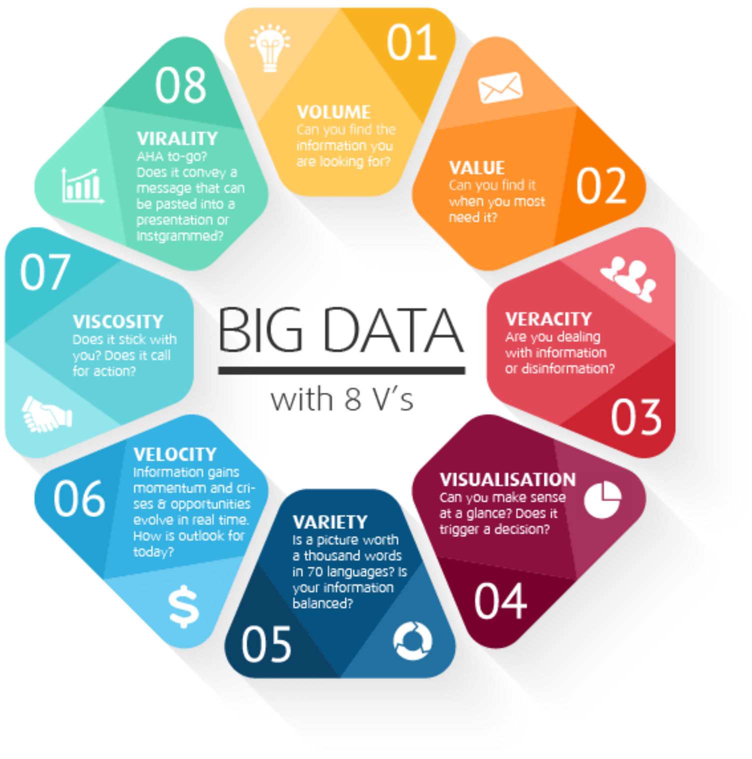

BIG DATA
with 8 V's
01
VOLUME
Can you find the information you are looking for?
02
VALUE
Can you find it when you most need it?
03
VERACITY
Are you dealing with information or disinformation?
04
VISUALISATION
Can you make sense at a glance? Does it trigger a decision?
05
VARIETY
Is a picture worth a thousand words in 70 languages? Is your information balanced?
06
VELOCITY
Information gains momentum and crises & opportunities evolve in real time. How is outlook for today?
07
VISCOSITY
Does it stick with you? Does it call for action?
08
VIRALITY
AHA to-go? Does it convey a message that can be pasted into a presentation or instgrammed?

Figure 11 – The 8 V's of Big Data [25]

Visualization – Big Data visualization involves the presentation of data of almost any type in a graphical format that makes it easy to understand and interpret. But it goes far beyond typical corporate graphs, histograms and pie charts to more complex representations like heat maps and fever charts, enabling decision makers to explore data sets to identify correlations or unexpected patterns [26].

Viscosity – Viscosity measures the resistance to flow in the volume of data. This resistance can come from different data sources, friction from integration flow rates, and processing required to turn the data into insight. Technologies to deal with viscosity include improved streaming, agile integration bus, and complex event processing [27].

Virality – Virality describes how quickly information gets dispersed across people to people (P2P) networks. Virality measures how quickly data are spread and shared to each unique node. Time is a determinant factor along with rate of spread [27].

Table 4 – The V's of Big Data [28]

#	Characteristics	Elucidation	Description
1	Volume	Size of Data	Quantity of collected and stored data
2	Velocity	Speed of Data	The transfer rate of data between source and destination
3	Variety	Type of Data	Different type of data like pictures, videos and audio arrives at the receiving end
4	Veracity	Data Quality	Accurate analysis of captured data is virtually worthless if it's not accurate
5	Value	Importance of Data	It represents the business value to be derived from Big Data
6	Visualization	Data Act/ Data Process	It is a process of representing abstract
7	Viscosity	Lag of Event	It is a time difference the event occurred and the event being described
8	Virality	Spreading Speed	It is defined as the rate at which the data are broadcast /spread by a user and received by different users for their use
9	Variability	Data Differentiation	Data arrives constantly from different sources and how efficiently it differentiates between noisy data or important data
10	Validity	Data Authenticity	Correctness or accuracy of data used to extract result in the form of information
11	Vulnerability	Data Security	Big Data brings new security concerns.
12	Volatility	Duration of Usefulness	Big Data volatility means the stored data and how long is useful to the user
13	Venue	Different Platform	Various types of data arrived from different sources via different platforms like personnel system and private & public cloud
14	Vocabulary	Data Terminology	Data terminology likes data model and data structures

#	Characteristics	Elucidation	Description
15	Vagueness	Indistinctness of existence	Vagueness concern the reality in information that suggested little or no thought about what each might convey
16	Verbosity	Data Redundancy	The redundancy of the information available at different sources
17	Voluntariness	Data Voluntariness	The will full availability of Big Data to be used according to the context
18	Versatility	Data Flexibility	The ability of Big Data to be flexible enough to be used differently for different context
19	Complexity	Correlation of Data	Data comes from different sources and it is necessary to figure out the changes whether small or large in data with respect to the previously arrived data so that information can get quickly

Table 4 shows many other examples of creativity of some authors in *V's race*. To close this race of defining more and more V's, look at the Figure 12, where the first appearances of these V's are symbolized.

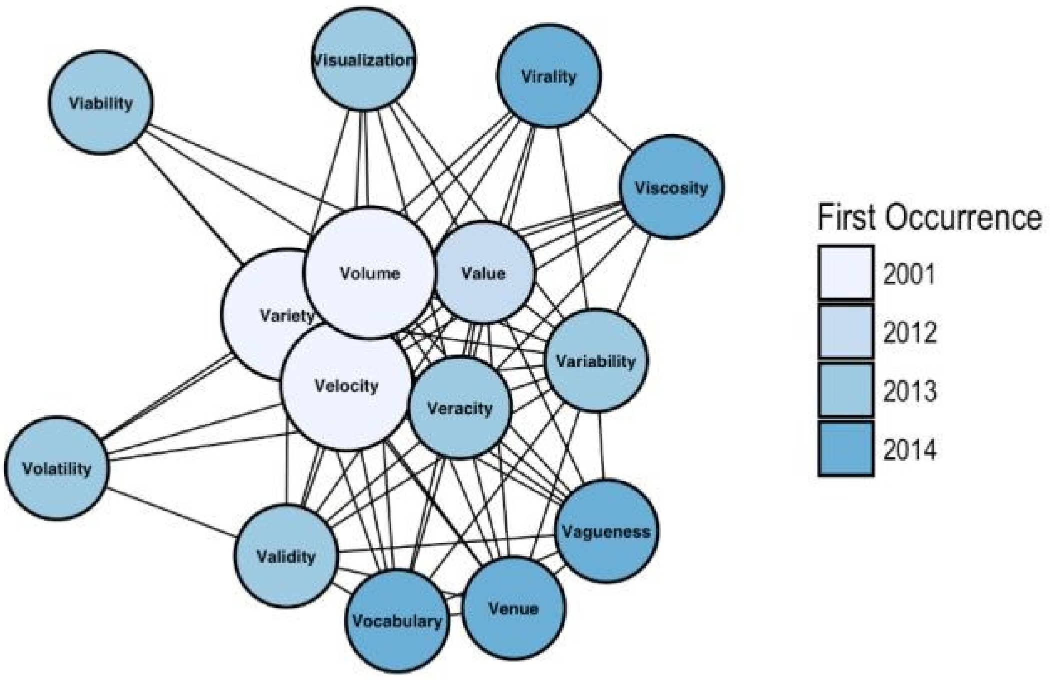


Figure 12 – First occurrence of V's by year [29]

1.8 Data Science vs. Big Data vs. Data Analytics

For companies, data are becoming more and more available, primarily due to the cost of acquisition, processing and storage. The problem is not how to obtain the data, but how to benefit of them. Every company has now huge amount of data and it brings new challenges how to utilize them, how to find real value inside. That's what will split them between successful and losers.

It was interesting to watch Gartner's Hype Cycle for Emerging Technologies for several last years. Big Data appeared in viewfinder of Gartner around 2010-11, in 2012-2013 reached top of hype cycle and in 2015 surprisingly disappeared with explanation, that Big Data is so pervasive that it can't really be considered an emerging technology any longer. But what appeared in Gartner's Hype Cycle in 2014? The answer could be found in Figure 13. It is a *Data Science* – the new area worth to follow.

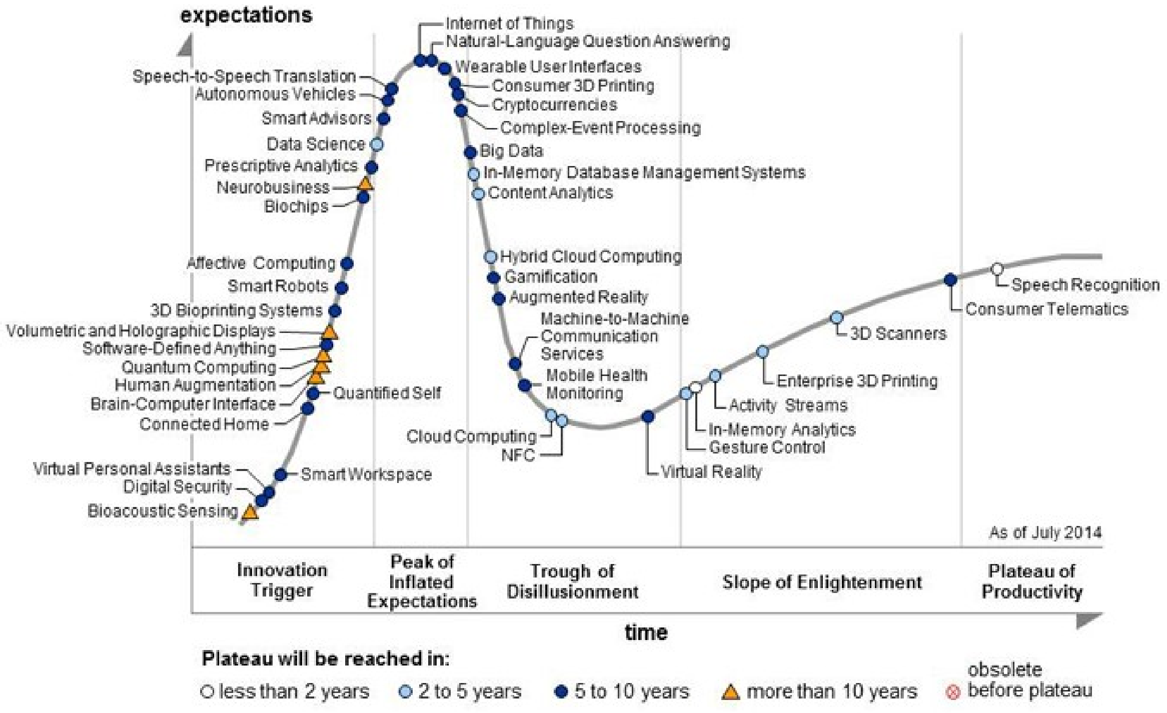


Figure 13 – Hype Cycle for Emerging Technologies, 2014 [30]

As it became common, there is always no proper definition, as everybody can see it from different perspective, everybody add something own. There are hundreds of definitions what

Data Science is or could be. Definitely the best definition could be found on Wiki page where the term Data Science is best described.

Data science is a multi-disciplinary field that uses scientific methods, processes, algorithms and systems to extract knowledge and insights from structured and unstructured data [31]. But the key word in data science is not "data", it is "science". Data science is only useful when the data are used to answer a question [32]. Data science is the same concept as Data mining and Big Data: "use the most powerful hardware, the most powerful programming systems, and the most efficient algorithms to solve problems" [33].

In 2012, when Harvard Business Review called it "The Sexiest Job of the 21st Century", [34] the term "data science" became a buzzword. It is now often used interchangeably with earlier concepts like business analytics, business intelligence, predictive modeling, and statistics. In many cases, earlier approaches and solutions are now simply rebranded as "data science" to be more attractive [35].

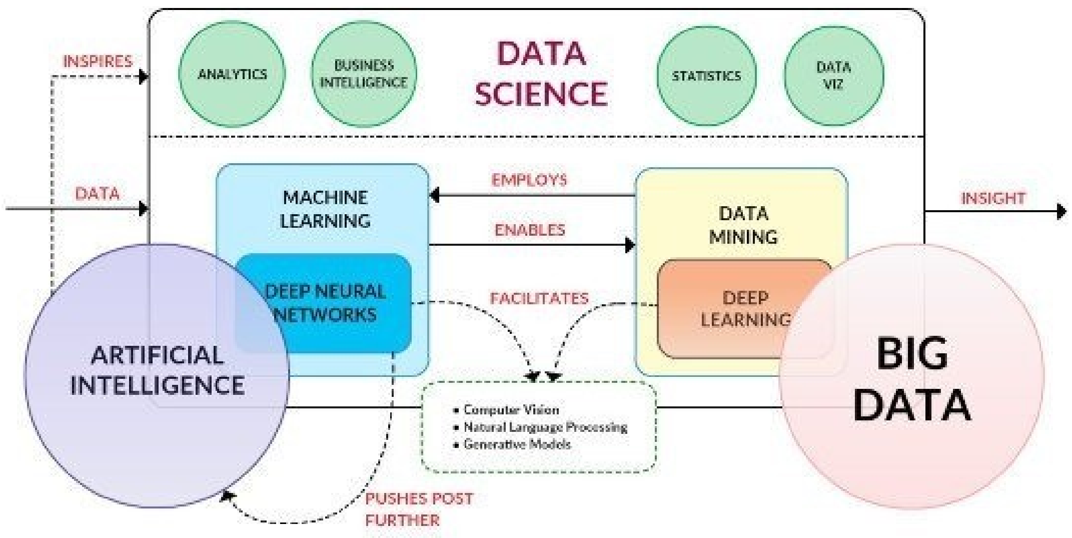


Figure 14 – Relations of Data Science with surrounding area [36]

It's very hard to explain the differences between Data Mining, Data Analytics, Data Analysis, Data Science and Big Data itself and how all fit together. Data Science helps in discovering useful information from Big Data for efficient data analysis with the help of

Artificial Intelligence and Machine Learning/Data Mining algorithms. Jonathan Nolis breaks data science down into three components:

(1) business intelligence, which is essentially about "taking data that the company has and getting it in front of the right people" in the form of dashboards and reports

(2) decision science, which is about "taking data and using it to help a company make a decision"

(3) machine learning, which is about "how can we take data science models and put them continuously into production." Although many working data scientists are currently generalists and do all three, we are seeing distinct career paths emerging, as in the case of machine learning engineers [37].

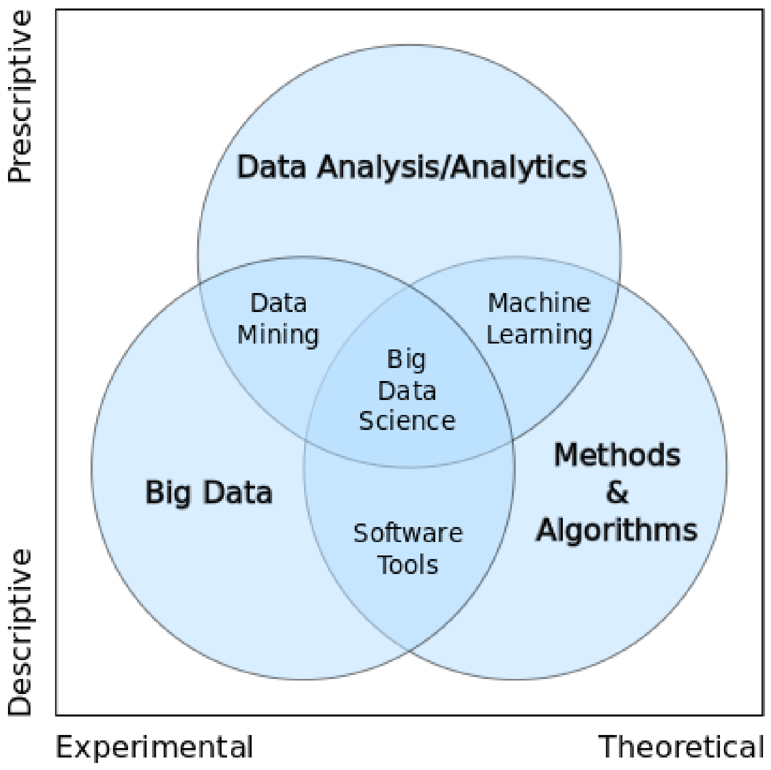


Figure 15 – The Fields of Data Science [38]

One or more pictures (Figure 14, Figure 15 and Figure 16) could say more than 100 words how all fit and relate together.

Data mining is analyzing data for the purpose of discovering unforeseen patterns or properties. It makes messy unstructured data into useful info. It is the computational process of discovering patterns in large data sets (involving Big Data abstraction) involving methods at the intersection of artificial intelligence, machine learning, and database systems. Data mining closely relates to data analysis. One can say that Data mining is data analytics operating on big data sets, because no small data sets would issue meaningful analytics insights. Data mining, shortly speaking, is the process of transforming data into useful information. Data mining is more rooted on the database (static, already stored data) point of view, whereas machine learning has been originated from a desire to make an Artificial Intelligence (AI). Algorithms used in Data mining: Apriori (finding associations), DBSCAN (finding clusters) and Decision trees.

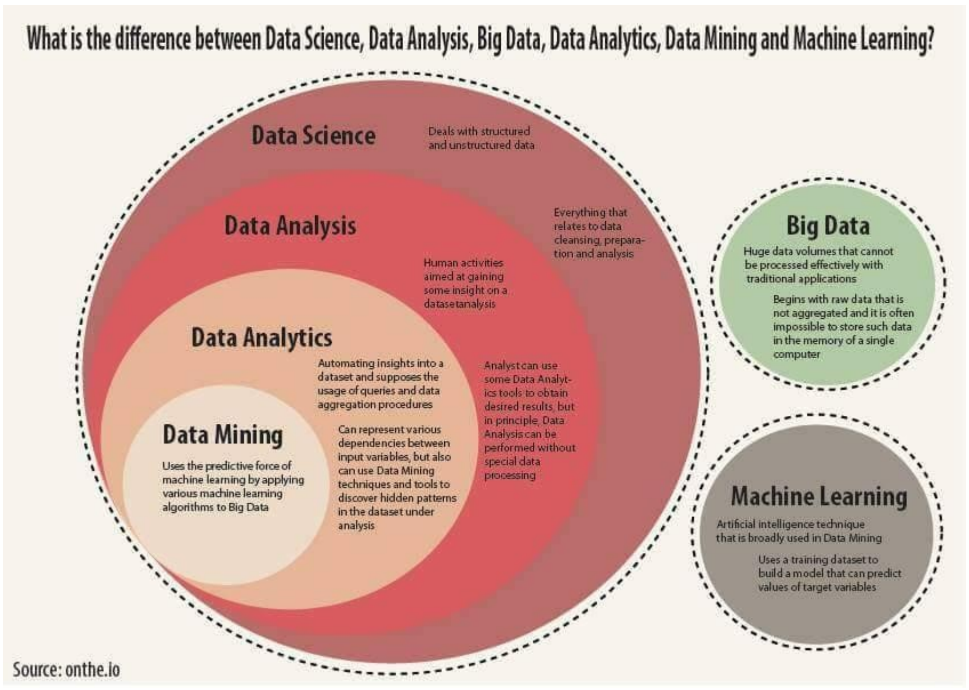

What is the difference between Data Science, Data Analysis, Big Data, Data Analytics, Data Mining and Machine Learning?
Data Science
Deals with structured and unstructured data
Everything that relates to data cleansing, preparation and analysis
Data Analysis
Human activities aimed at gaining some insight on a datasetanalysis
Analyst can use some Data Analytics tools to obtain desired results, but in principle, Data Analysis can be performed without special data processing
Data Analytics
Automating insights into a dataset and supposes the usage of queries and data aggregation procedures
Can represent various dependencies between input variables, but also can use Data Mining techniques and tools to discover hidden patterns in the dataset under analysis
Data Mining
Uses the predictive force of machine learning by applying various machine learning algorithms to Big Data
Big Data
Huge data volumes that cannot be processed effectively with traditional applications
Begins with raw data that is not aggregated and it is often impossible to store such data in the memory of a single computer
Machine Learning
Artificial intelligence technique that is broadly used in Data Mining
Uses a training dataset to build a model that can predict values of target variables
Source: onthe.io

Figure 16 – Data Science [39]

Data Analysis is a heuristic activity, where scanning through all the data the analyst gains some insight (makes useful information). Data Analysis leverages statistical methods to analyze aggregated or non-aggregated data.

Data Analytics is about applying a mechanical or algorithmic process to find insights. For example, running through various data sets with a purpose of finding meaningful correlation between them. This takes the use of statistics and data science tools. Analytics are the result of analysis and the form of presentation of the analysis results; might simply prediction interest.

Machine Learning finds patterns in (big) data that useful for researchers and are not visible from human point of view. Machine Learning is not a static, hard-coded model but a self-learning, self-adjusting model (machine learns and changes itself). Machine Learning or Artificial Intelligence compared to Data Mining is more on incorporating acquired knowledge into the framework for further (i.e. future) use in analysis.

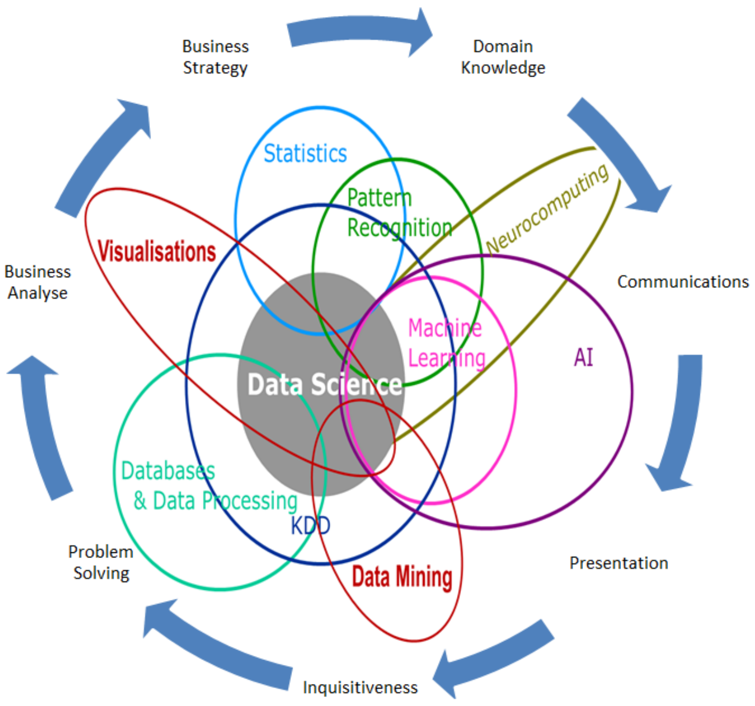


Figure 17 – Data Science Is Multidisciplinary [40]

Data Science is a science field that includes methods and processes for operating over data. It’s a cluster of mathematics, statistics, programming, and ingenious ways of capturing data that may not be being captured right now. Data Science includes Machine Learning and other methods like problem formulation, exploratory data analysis, data model compiling, data visualization, data extraction, etc. [38].

As you can see, there are many views from different perspective on Data Science, Data Analytics, Data Analysis, Data Mining, Machine Learning, Big Data and other popular term used in the industry these days. It would be beneficial to finish this subchapter with last picture (see Figure 17) of which all areas Data Science touch. There are so many discipline involve, that we have to wait few years to let it settled down.

1.9 Application of Big Data

Listing all areas where Big Data are used today would be a superhuman task. In the last 10 years, this phenomenon has come to all areas of human activity. In the following paragraph are presented some of the application of Big Data technology by the industry [41].

Retail/Consumer

- Merchandizing and market basket analysis
- Campaign management and customer loyalty programs
- Supply-chain management and analytics
- Event- and behavior-based targeting
- Market and consumer segmentations

Finances & Frauds Services

- Compliance and regulatory reporting
- Risk analysis and management
- Fraud detection and security analytics
- Credit risk, scoring and analysis
- High speed arbitrage trading
- Trade surveillance
- Abnormal trading pattern analysis

Web and Digital media

- Large-scale clickstream analytics
- Ad targeting, analysis, forecasting and optimization
- Abuse and click-fraud prevention
- Social graph analysis and profile segmentation
- Campaign management and loyalty programs

Health & Life Sciences

- Clinical trials data analysis
- Disease pattern analysis

- Campaign and sales program optimization
- Patient care quality and program analysis
- Medical device and pharmacy supply-chain management
- Drug discovery and development analysis

Telecommunications

- Revenue assurance and price optimization
- Customer churn prevention
- Campaign management and customer loyalty
- Call detail record (CDR) analysis
- Network performance and optimization
- Mobile user location analysis

Ecommerce & customer service

- Cross-channel analytics
- Event analytics
- Recommendation engines using predictive analytics
- Right offer at the right time
- Next best offer or next best action

To at least partially present Big Data in detail, use case from the automotive industry was chosen and are presented in following paragraphs.

Automotive Industry Use Case of Big Data

The average vehicle on today's roads generates huge amounts of data. Vehicle sensors monitor everything from tire pressure, engine speed to oil temperature and vehicle speed or position. Thus, vehicles can produce anywhere from 5 to 250 GB of data per hour. The study from 2014 by McKinsey states that the average connected car generates 25 GB of data per hour [42]. Patrick Nelson from IDG, says a self-driving car can generate up to 4 TB of data per day (details are depicted in Figure 18) [43]. Autonomous vehicles, such as Google, generate about 1 GB of data every second. The vast majority of this data are used in real time to check and report on vehicle functions and has no real long-term value. Taking thousands of messages from the sensor saying "Normal tire pressure" does not bring any further benefit, so car makers do not bother to store this data in the car or on servers. However, some data are valuable. And if such information can be gathered from a substantial portion of the billions of cars in operation today, we do not need more than basic knowledge of arithmetic to understand why Big Data is attracting so much attention in the automotive sector.

There is a great interest in collecting large data not only by OEMs (Original Equipment Manufacturer), but also by companies that have nothing to do with OEM production or delivery. These are companies that offer solutions to the surrounding ecosystem. Big Data opens up many other possibilities. Another example how to use large amounts of data is to understand how customers actually use their products. In most cars, some features are unnoticed while others are highly valued. With large amounts of data, the OEM has the opportunity to learn a lot more about how customers use the vehicle and what their preferences are. By analyzing data, the data collected can be used to plan and develop other features.

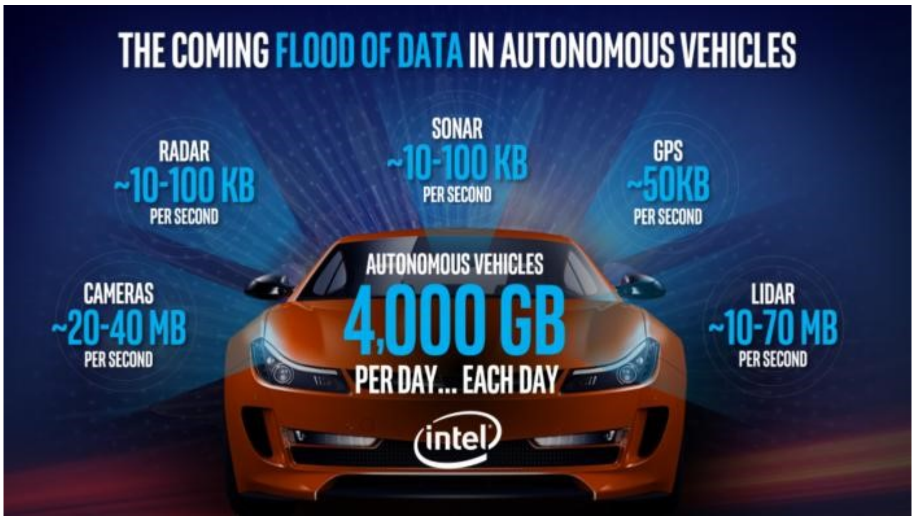

THE COMING FLOOD OF DATA IN AUTONOMOUS VEHICLES
RADAR ~10-100 KB PER SECOND
SONAR ~10-100 KB PER SECOND
GPS ~50KB PER SECOND
CAMERAS ~20-40 MB PER SECOND
AUTONOMOUS VEHICLES 4,000 GB PER DAY... EACH DAY
LIDAR ~10-70 MB PER SECOND
intel

Figure 18 – The coming flood of data in autonomous vehicles [43]

It is useful for the automaker to monitor the habits of a particular customer group, the number of their short trips on a single day or long rides on weekends, or the frequency of rides in difficult winter conditions. In addition, manufacturers can begin to focus on additional value-added services that can target specific demographic groups within their customer base.

Interesting customers can be insurance companies that want to collect a particular subset of car data - such as speed, driving time, braking and cornering - to find out how drivers actually drive, rather than base their risk calculation on much less reliable sources like their age or credit status. Insurance companies are already on the market with replacement solutions that enable this data collection. An example of this is the AXA Drive application, which is able to record driver's driving style with GPS and gyro sensors. Patiently collects data, records abrupt starts, passages of dangerous places, taking into account day, night, meteorological conditions and other details. After driving, it offers tips for improvement and greater safety, but how the data are utilize internally, there's no evidence. But we can be sure, they can score individual drivers and offer them better insurance rate. More and more vehicle manufacturers are also offering insurance services through their partners, so there are more data seekers who can provide them with sufficient details to calculate the risk.

Another example of how to profit from Big Data is that manufacturers can sell their data to third parties. For example, USPS has priced for its large data and allows organizations to access the National Address Change Database for $ 175,000 a year. Cisco has recently calculated that the financial benefits of large data for vehicle manufacturers will be close to $300 per vehicle per year. Most of this amount comes from lower warranty service costs and improved concept design. In the same analysis, it was concluded that large data would help drivers save $500 a year thanks to better navigation and smarter driving.

To collect data for car manufacturers, it is now necessary to wait for the driver to arrive with their cars at a service whose technicians have access to the data through the on-board diagnostic system. But not all drivers have their cars settled regularly, for example, 15,000 km, which means that data analysis from all the vehicles in operation is not going on in a continuous and proactive way, and achieving this is still a challenge. With the arrival of cars connected to the Internet respectively, using 5G mobile networks, car makers can have full access to data from the car anywhere, anytime, making the analysis of this data easier and more viable. But as long as these still connected cars do not make up the majority of the cars they operate, the *big data* will be rather *small data* [44] [45].

2 NOSQL

2.1 Introduction

The main characteristics of NoSQL DBMS are flexible scalability, lower cost, flexible data structure and availability. The capacity of the cluster is much easier to expand with horizontal scaling (involves running multiple servers (nodes) within a cluster). In contrast vertical scaling, data migration or at least shutdown of the system is necessary in case of capacity increase. NoSQL databases are designed to allow new nodes to be easily added or removed as needed without shutdown. High availability can be assured by other nodes taking on the role of individual nodes in the event of failure or maintenance. A significant cost factor is the fact that commodity hardware can be used as clustered servers, which is significantly less expensive than large database servers. Vast majority of NoSQL databases are available as open source products with the option to purchase support as commercial services provided by third parties. Almost all NoSQL databases offer the ability to represent data without a fixed data schema. As mentioned above, the main reason for interest in NoSQL databases is the ability to run databases in clusters. Whether the database system will perform optimally, for example, with a large number of write operations or whether it will guarantee availability in the event of disruption between servers, depends on the distribution model be chose.

2.2 Distribution

NoSQL databases, also known as distributed databases, are based on the principle of data distribution, as the name implies. However, this does not mean that data distribution is necessary when using the NoSQL database. If the size of the data allows it, only a minimum to no data distribution can occur, what limit or eliminate the risks associated with the distribution. The basic model is running a database system on a single server. Even in this case, it may be worthwhile to use a NoSQL DBMS, if appropriate from the perspective of the data structure that the application is working with. The following data distribution techniques are used for ideal data processing.

First distribution model is called *sharding*, i.e. placing different parts of data on different nodes. This approach is useful when different users access different parts of the data. Many NoSQL DBMS supports *auto-sharding*, where the database system itself allocates data to individual nodes. Sharding is a convenient way to simultaneously increase both read and write performance.

Another distribution model is *master-slave replication*. The data are replicated to several nodes, one of which is designated as a primary node (called *master*) and is generally responsible for executing data update operations. The replication mechanism then synchronizes the secondary nodes (called *slaves*) with the current data on the primary node. Master-slave replication is useful when a large number of read operations are performed on the data. Conversely, the Peer-to-Peer replication distribution model can help when data are written frequently. There is no primary node, all nodes have the same weight and perform read and write operations. Replication is performed between nodes. The downside of this model is the risk of data violation. This occurs when several write operations attempt to modify the same entry on different nodes. This problem can be solved by mutual coordination between nodes, but this is associated with higher network traffic intensity. In this case, it is not necessary to wait for the response of all nodes, but to ensure a high degree of consistency, their absolute majority is sufficient. The number of nodes to receive confirmation is called a *quorum*. The second possible solution is to allow the emergence of such conflicts of enrollment operations and to subsequently address them within the application in accordance with its business logic.

2.3 Consistency

Efficient and correct data processing requires consistency. Therefore, traditional, relational databases require that the data contained therein meet certain restrictive conditions, so-called *integrity constraints*. These database systems work with so-called *transactions* that are logically related operations that convert data from one consistent state to another. At the end of the transaction, all integrity restrictions must be met. For this reason, the following features are required for transactions that are referred to as ACID transaction. An important feature of NoSQL databases is the absence of a classic transaction approach and a different view of data consistency. While traditional RDBMS is based on the principle of ACID transactions (**A**tomicity, **C**onsistency, **I**solation, **D**urability), the principle of NoSQL databases is BASE (**B**asic **A**vailability, **S**oft state, **E**ventual consistency).

Atomicity - the database transaction is as an operation further indivisible (atomic). It is performed either as a whole or not at all (and the database system gives the user a note, e.g. by an error message).

Consistency - no integrity constraints are violated after the transaction. Database status after transaction completion is always consistent, i.e. valid according to all defined rules and restrictions. There is never a situation where the database is in an inconsistent state.

Isolation - operations inside a transaction are hidden from external operations. Returning a transaction (ROLLBACK operation) does not affect another transaction, and if so, it must be returned. This behavior may result in a cascading rollback.

Durability - changes made as a result of successful transactions are actually stored in the database and can no longer be lost.

The term "**B**asic **A**vailability" means if single node fails, part of the data won't be available, while the remaining parts of the system remain functional. "**S**oft state" refers to the fact that data may be overwritten by a newer version, which is closely related to the third attribute of the BASE principle - "**E**ventual consistency". This term means that the database may, under certain circumstances, be in an inconsistent state. Typically, this is where copies of the same data are located on several nodes within a cluster. If a user or application updates data on one of the servers, other copies of the data are inconsistent for a certain (usually short) period until the NoSQL database replication mechanism updates all copies of the data.

Unlike ACID, BASE is not so strict and permits temporary data inconsistency to increase availability and performance. For BASE access, the database is also available in case of partial failures between nodes, or at least parts of it (e.g. in case of node failure). This results in temporary data inconsistency. This also occurs with delays in synchronizing changes between individual parts. However, if the connection is restored and the database is unloaded for some time, a consistent state is restored. Since it does not block writing (as opposed to ACID transactions), it is more responsive, even at the cost of temporary inaccuracies in stored data.

2.4 CAP Theorem

CAP theorem also known as Brewer theorem was introduced by Eric Brewer in 2000 at *ACM Symposium on Principles of Distributed Computing* and expresses a triple constraints related to distributed database systems. It states that a distributed database system, running on a cluster, can only provide two of the following three properties:

Consistency – a read from any node results in the same data across multiple nodes. A guarantee that every node in a distributed cluster returns the same, most recent data. Consistency refers to every client having the same view of the data. There are various types of consistency models. Consistency in CAP (used to prove the theorem) refers to linearizability or sequential consistency, a very strong form of consistency.

Availability – a read/write request will always be acknowledged in the form of a success or a failure. Every non-failing node returns a response for all read and write requests in a reasonable amount of time. The key word here is *"every"*. To be available, every node on (either side of a network partition) must be able to respond in a reasonable amount of time.

Partition tolerance – the database system can tolerate communication outages that split the cluster into multiple silos and can still service read/write requests. The system continues to function and upholds its consistency guarantees in spite of network partitions. Distributed systems guaranteeing partition tolerance can gracefully recover from partitions once the partition heals.

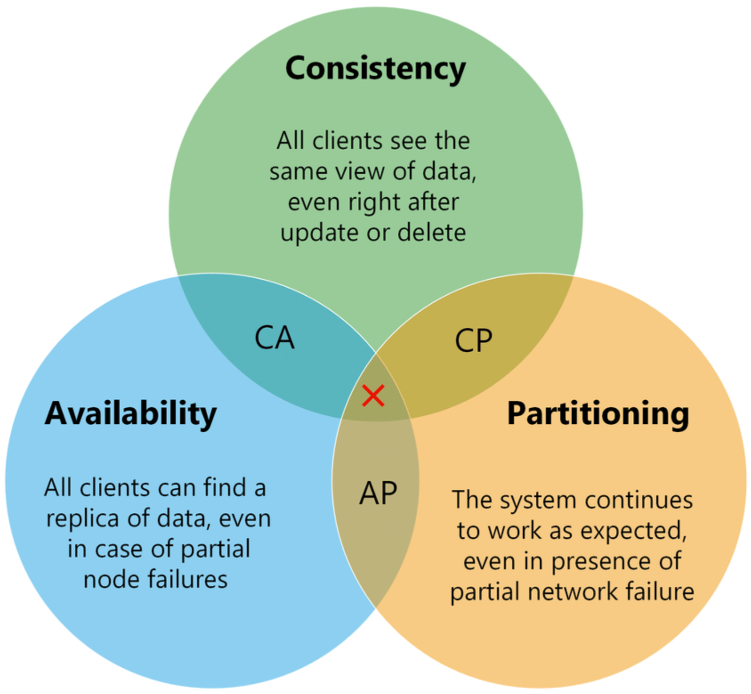

Consistency
All clients see the same view of data, even right after update or delete
CA
CP
Availability
All clients can find a replica of data, even in case of partial node failures
Partitioning
AP
The system continues to work as expected, even in presence of partial network failure

Figure 19 – A Venn diagram summarizing the CAP theorem

In 2002, Seth Gilbert and Nancy Lynch of MIT published a formal proof of Brewer's conjecture [46]. The theorem states that networked shared-data systems can only guarantee/strongly support two of the three properties. The CAP theorem categorizes systems into three categories:

CP (Consistent and Partition Tolerant) - At first glance, the CP category is confusing, i.e., a system that is consistent and partition tolerant but never available. CP is referring to a category of systems where availability is sacrificed only in the case of a network partition.

CA (Consistent and Available) - CA systems are consistent and available systems in the absence of any network partition. Often a single node's DB servers are categorized as CA systems. Single node DB servers do not need to deal with partition tolerance and are thus considered CA systems. The only hole in this theory is that single node DB systems are not a network of shared data systems and thus do not fall under the preview of CAP.

AP (Available and Partition Tolerant) - These are systems that are available and partition tolerant but cannot guarantee consistency.

Figure 19 – A Venn diagram summarizing the CAP theorem shows the overlapping areas between Consistency, Availability and Partition tolerance.

However, the CAP theorem cannot be taken literally, and its critics point out a few points for discussion. Above all, it is the fact that the originally used definitions are not entirely accurate and the particular theorem, which has been proven later, has other restrictive conditions. Furthermore, some properties are questionable. E.g. while the lack of consistency is assumed throughout the system's work, the lack of availability is only in the case of network disconnection, that is, sometimes only. Thus, these two conditions do not have symmetrical properties. Or if we understand the theorem literally, then it says that providing a combination of consistency conditions and resistance to network disintegration means that the system is not available at all after disconnection. This is, of course, a very bad feature [47] [48] [49].

2.5 Classification of Database Management Systems

Before the basic types or classification of NoSQL databases will be presented, it is necessary to step back and look at database classification from higher perspective and present top most DBMS classification from different edges. There are several criteria based on which DBMS are classified.

Based on the data model

Relational database - definitely most popular data model used worldwide. It is based on the SQL and ACID transaction paradigm (**A**tomicity, **C**onsistency, **I**solation, **D**urability). A relational database is a set of formally described tables from which data can be accessed or reassembled in many different ways without having to reorganize the database tables. The tables or the files with the data are called as relations that help in designating the row or record, and columns are referred to attributes or fields. Examples: Oracle, MySQL. Microsoft SQL Server.

Object oriented database - object oriented database management systems (often referred to as object databases) were developed in the 1980s motivated by the common use of object-oriented programming languages. The goal was to be able to simply store the objects in a database in a way that corresponds to their representation in a programming language, without the need of conversion or decomposition. Examples: InterSystems Caché, Versant Object Database, Db4o.

Hierarchical database - in which the data are organized into a tree-like structure. The data are stored as records which are connected to one another through links. A record is a collection of fields, with each field containing only one value. The type of a record defines which fields the record contains. Examples: IMS (IBM), Windows registry (Microsoft).

Network database – is a database model similar to a hierarchical database model that has been almost exclusively used by the database model for a long time. In addition to the hierarchical database model, it provides more to more relationships, so one entity could have more parents. However, this data concept was overcome in 1970 by the relational database concept. In addition, it also allows recursion, i.e. the entity can be the parent of its parent. The disadvantage of using a network database is its inflexibility and the resulting difficult change in its structure. Examples: RDM Server, Integrated Data Store (IDS).

Based on the number of users

Single user - supports only one user at one point of time. It is mostly used with the personal computer on which the data resides accessible to a single person.

Multiple users - supports two or more simultaneous users concurrently. Data can be both integrated and shared, a database should be integrated when the same information is not need be recorded in two places.

Based on the distribution

Centralized database system - keeps the data in one single database at one single location. In a centralized database system, a single machine called a database server hosts the DBMS and the database.

Distributed database system – here, data and the DBMS software are distributed over several sites but connected to the single computer. Main difference between centralized and distributed database systems is, the data resides in several locations or on multiple servers at the same location.

Parallel network database system - the advantage of improving processing input and output speeds is in use of multiple processors such as cluster server that host the DBMS. Majorly used in the applications that have query to larger database. It holds the multiple central processing units and data storage disks in parallel.

Client-server database system - has two logical components namely client and server. Clients are generally the personal computers or workstations whereas servers are the large workstations, mini range computers or a main frame computer system.

Based on other criteria

There are many more classifications of DBMS such

- Based on cost (Low cost, Medium cost, High cost DBMS)
- Based on access (Sequential access, Direct access, Inverted file structure)
- Based on usage (OLTP – Online Transaction Processing, OLAP – Online Analytical Processing, Big data and analytics DBMS, XML, Multimedia, GIS, Sensor, Mobile, Open Source .. and many others)

NoSQL Databases Classification

Regarding the NoSQL databases, the classification would be possible depending on which two properties of the CAP theorem the particular database system primarily focuses on. For most NoSQL databases, there is a possibility of one of two solutions, either the database system favors properties CP, or prefers the properties AP. For better visualization how different type of NoSQL databases fit to AC, AP or CP group, please see Figure 20.

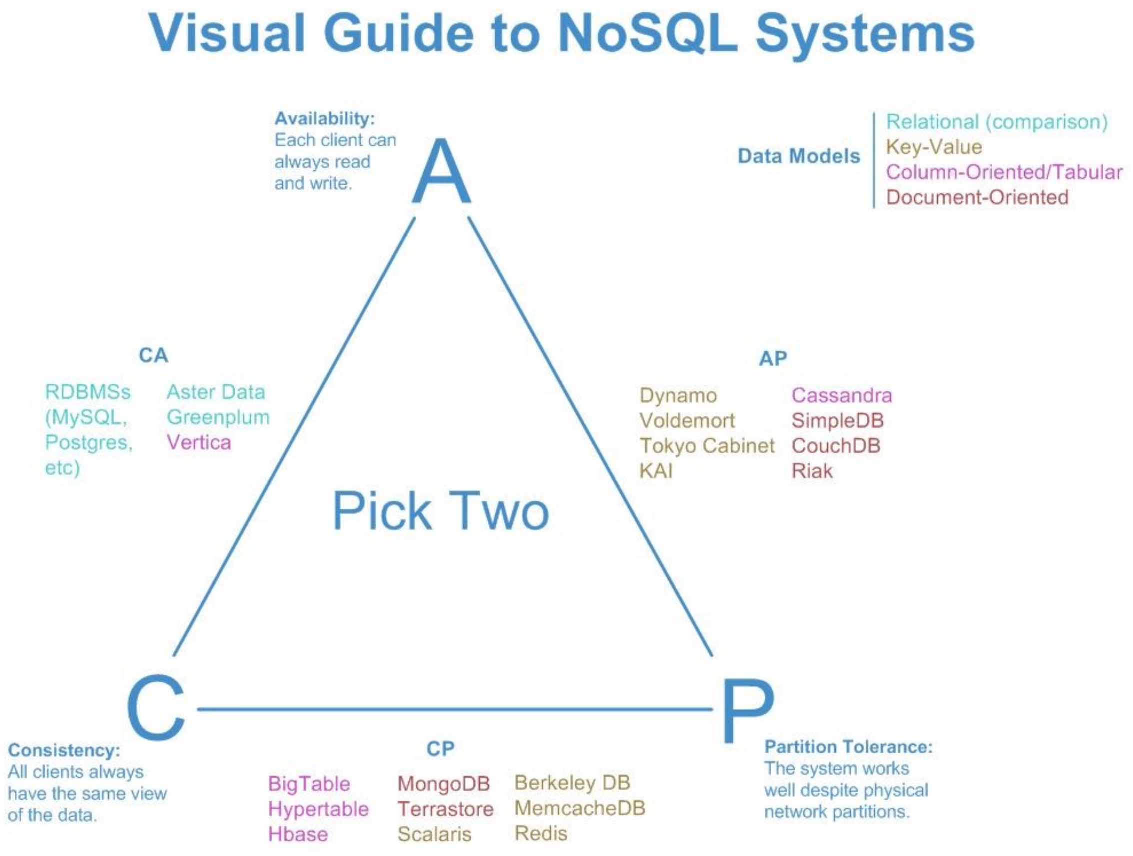


Figure 20 – Example of NoSQL databases by two of CAP [50]

According to the CAP Theorem, you can only pick two.

- Consistency means that each client always has the same view of the data.
- Availability means that all clients can always read and write.
- Partition tolerance means that the system works well across physical network partitions.

CP (Consistent, Partition-Tolerant) - systems have trouble with availability while keeping data consistent across partitioned nodes. Examples of some of the databases belonging into CP systems are listed in the Table 5.

Table 5 – Examples of CP

Database Name	Database Type
Bigtable	column-oriented/tabular
Hypertable	column-oriented/tabular
HBase	column-oriented/tabular
MongoDB	document-oriented
Terrastore	document-oriented
Redis	key-value
Scalaris	key-value
MemcacheDB	key-value
Berkeley DB	key-value

AP (Available, Partition-Tolerant) - Systems achieve "eventual consistency" through replication and verification. Examples of some of the databases belonging into CP systems are listed in the Table 6.

Table 6 – Examples of AP

Database Name	Database Type
Dynamo	key-value
Voldemort	key-value
Tokyo Cabinet	key-value
KAI	key-value
Cassandra	column-oriented/tabular
CouchDB	document-oriented
SimpleDB	document-oriented
Riak	document-oriented

The next classification could be based on the method of querying. However, most NoSQL databases offer several ways how to query their data.

A high-level taxonomy of the NoSQL datastores based on the data model can classify them into five major categories: key-value stores, document stores, wide-column (column-oriented) stores, graph databases and multi-model databases. This classification is rather common and we describe each of them in more detail in following subchapters.

2.6 Key Value Databases

Key value databases (stores) are the simplest form of NoSQL databases and are essentially hash tables. From the perspective of the application's interaction with the database system, Key-Value database solutions are characterized by a very straightforward way of storing and reading data. The client application can read the value based on a particular key, save a value under the key, or delete the key along with the value from the database.

Unlike relational databases, key-value databases do not contain tables or any equivalent. Some database systems of this type support the creation of multiple namespaces within a single database. An important feature of NoSQL key-value databases, which comes from their very nature, is the fact that, unlike relational databases and other types of NoSQL databases, it is not possible to search for data according to their value. Data are always accessed via a key.

The advantage of these databases is the high speed and the fact that they can be easily distributed. The negative of key-value database systems is that they cannot be searched for by stored values.

Key-value databases are very simple, mean by their structure or operations on data. For all systems of this type, only three basic operations are common, which are - to *insert* a value for a given key (PUT), *obtain* a value for a given key (GET), and the last is to *delete* a key-value pair (DELETE). Some systems of this type also allow for more complex operations, but this foundation is common to all and sufficient to handle data in many applications.

Table 7 – DB-Engines Ranking of Key-value stores [51]

Rank	DBMS	Database Model	Score	Link
1.	Redis	Key-value, Multi-model	146.38	[52]
2.	Amazon DynamoDB	Multi-model	56.01	[53]
3.	Memcached	Key-value	28.73	[54]
4.	CosmosDB	Multi-model	26.28	[55]
5.	Hazelcast	Key-value	8.35	[56]
6.	Ehcache	Key-value	6.49	[57]
7.	Aerospike	Key-value	6.21	[58]
8.	OrientDB	Multi-model	6.19	[59]
9.	Riak KV	Key-value	5.70	[60]
10.	Ignite	Multi-model	5.09	[61]

To choose the best or most used database in each category is un-human task, each of particular database has its pros/cons and for individual project or usage it's necessary to weight them. For our purpose will be used the most recognized and reputed list of ranked database by their popularity from site db-engines.com (see Table 7).

2.7 Document Databases

Document databases are more complex structures than key-value databases. Databases of this type are suitable for storing and managing documents. The document in this case represent a data structure with a self-descriptive character of the data. This means that in addition to the data itself, the documents also contain metadata that describes the meaning of the data. Unlike simple key-value storage, document databases allow content-based searches.

Individual documents stored in this type of database are usually encoded in XML or JSON formats, or binary form of JSON called BSON (Binary JSON). JSON (JavaScript Object Notation) is a text format for data exchange. It represents four simple data types (string, number, logical value, and null) and two structured types, which are object and array. Structured data types can include any other arbitrary type, whether simple or structured. As a result, JSON is a very flexible data format where almost any data can be written.

Table 8 – DB-Engines Ranking of Document stores [51]

Rank	DBMS	Database Model	Score	Link
1.	MongoDB	Document	401.98	[62]
2.	DynamoDB	Multi-model	56.01	[53]
3.	Couchbase	Document	36.28	[63]
4.	Cosmos DB	Multi-model	26.28	[55]
5.	CouchDB	Document	20.44	[64]
6.	MarkLogic	Multi-model	14.47	[65]
7.	Firebase Realtime Database	Document	11.00	[66]
8.	OrientDB	Multi-model	6.19	[59]
9.	RavenDB	Document	4.66	[67]
10.	Google Cloud Datastore	Document	4.43	[68]

The key feature of document databases is the creation and use of search indexes. For example, MongoDB automatically creates indexes at the _id level within all collections. It also allows you to create a search index on any other value, including nested values. It is generally recommended that each attribute that is expected to be searched for in the future is assigned a custom index. If the index does not exist for the selection criteria in a given collection, the database does a full table scan as well as in relational databases. This is obviously undesirable in terms of time and unnecessary extraction of the I/O operations. In the following Table 8 are again listed most used and popular Document Stores.

2.8 Column Base Databases

Column-oriented (column based, columnar, wide column) databases are probably the most complex type of NoSQL databases in terms of basic structural elements. Some of the terms known from relational databases, such as column and row, are commonly used for this type of database, but their meaning is not identical and the differences with RDBMS are explained below. The basic element for storing data in column-oriented databases is a column. The column consists of a column name and a value. Some column-based database systems, along with a column and value, also maintain a timestamp. The row then consists of a set of specific columns. Each row of one column family can contain different columns. Similar to document databases, column-oriented databases do not require a predefined fixed schema, and columns can be arbitrarily added at runtime. Thus, although the data model structure may not be pre-designed in complete and definite form, and any columns can be added at any time while the application is running, the significant limitation is that the future performance and performance of the database depend on the high-quality and sophisticated design of the data model. Column-oriented databases are designed to support rows with a large number of columns and database systems supporting several million columns are no exception. Although column-oriented NoSQL databases may seem at first glance very similar to relational databases, they differ significantly in the data models and implementation method used. The most important difference is that column-oriented databases do not support JOIN operations. This is why data are stored in denormalized form in column-oriented databases. For data manipulation, they use SQL-like database-based database systems that support basic statements such as SELECT, INSERT, UPDATE, and DELETE, but do not contain, for example, any similar JOIN statement. As column-oriented databases do not use JOIN operations, thus have very good scalability across distributed systems and enable efficient data replication. In the following Table 9 you can find most used and popular Wide Column Stores.

Table 9 – DB-Engines Ranking of Wide column stores [51]

Rank	DBMS	Database Model	Score	Link
1.	Cassandra	Wide column	123.61	[69]
2.	HBase	Wide column	58.66	[70]
3.	Cosmos DB	Multi-model	26.28	[55]
4.	Datastax Enterprise	Wide column, Multi m.	9.17	[71]
5.	Microsoft Azure Table Storage	Wide column	4.50	[72]
6.	Accumulo	Wide column	4.12	[73]
7.	Google Cloud Bigtable	Wide column	1.81	[74]
8.	ScyllaDB	Wide column	1.62	[75]
9.	MapR-DB	Multi-model	0.75	[76]
10.	Alibaba Cloud Table	Wide column	0.21	[77]

2.9 Graph Databases

Another type of NoSQL database, graph databases are the last of 4 basic types of NoSQL databases. They can be considered as a special type of NoSQL databases, suitable only for data with a large number of mutual links. Graph databases allow you to store entities together with relationships between these entities. Entities form graph nodes and have properties. Relationships between entities are represented by oriented edges. Relationships between entities can also be modeled in relational databases; a typical example could be employee-boss relationship. The difference, however, is that if a representation of another similar relationship is added to the relational database, it is generally necessary to make significant changes to the data model. Graph databases are, unlike RDBMS, suitable for implementation requiring multi-level graph browsing, not just searching for links between two adjacent entities. Graph databases allow you to model relationships not only between domain entities, but also relationships that allow, for example, to sort entities into certain categories or link lists sorted by attributes. The number of links an entity can have is not limited in principle. The search for entities, i.e. graph nodes, is done by specifying the edge orientation, the property that the edge represents and the node from which the search should start. It is also possible, for example, to query the shortest path between two nodes.

Unlike other NoSQL databases, it is very difficult to perform data scaling in graph databases, i.e. to disperse data representing a single graph into multiple servers. This is due to the close connection of the nodes within the graph. However, data replication can be performed without problems, allowing better data reading performance. However, deploying write operations to a database located on multiple servers can be difficult. In the Table 10 are again listed most used and popular Graph databases.

Table 10 – DB-Engines Ranking of Graph databases [51]

Rank	DBMS	Database Model	Score	Link
1.	Neo4J	Graph	49.49	[78]
2.	Cosmos DB	Multi-model	26.28	[55]
3.	OrientDB	Multi-model	6.19	[59]
4.	ArangoDB	Multi-model	4.29	[79]
5.	Virtuoso	Multi-model	3.31	[80]
6.	Amazon Neptun	Graph	1.39	[81]
7.	JanusGraph	Graph	1.38	[82]
8.	Giraph	Graph	1.20	[83]
9.	Dgraph	Graph	1.08	[84]
10.	GraphDB	Multi-model	0.97	[85]

2.10 Multi-model Databases

In recent years, an enormous amount of new NoSQL databases appeared on the market to address the *Variety* of Big Data. Most of them has been organized around a single data model that determines how data can be organized, stored, and manipulated. In the coming years, as companies began using these databases, they also began to realize that they had a different databases specialized for different model of data or project supported by developers with unique knowledge. Thus, there was a pressure on the developers of these new databases to embed support in their already established databases for a different type of data. So, the so-called multi-model NoSQL databases began to appear on the market, combining the support of two or more types. Multi-model databases are designed to support multiple data models against a single, integrated backend. In these days, most of the most used NoSQL databases incorporated support for other data model, so it’s hard to present any list with ranked by popularity. Anyway, to mention some of them, among multi-model databases are generally considered these from the Table 11 (in alphabetic order) [86].

Table 11 – Multi-model databases [86]

Database Name	Database Type
ArangoDB	document (JSON), graph, key-value
Cosmos DB	document (JSON), key-value, SQL
Couchbase	document (JSON), key-value, N1QL
Datastax	key-value, tabular, graph
EnterpriseDB	document (XML and JSON), key-value
MarkLogic	document (XML and JSON), graph triplestore, binary, SQL
Oracle	relational, document, graph triplestore, property graph, key-value,
OrientDB	document (JSON), graph, key-value, reactive, SQL
Redis	key-value, document (JSON), property graph, streaming, time-series
SAP HANA	relational, document (JSON), graph, streaming

II. ANALYSIS

3 APACHE HADOOP ECOSYSTEM

In practical part of this work, most popular and used technology of Big Data – Apache Hadoop will be presented. First, core components of Hadoop itself like HDFS and YARN will be introduced along with step-by-step installation and configuration instructions. Presentation of technology will continue with MapReduce principles and practical examples how to use it. In the second half of practical part, most of the significant surrounding projects attached to Apache Hadoop will be shortly presented, again with installation and configuration instructions and several examples how to use them.

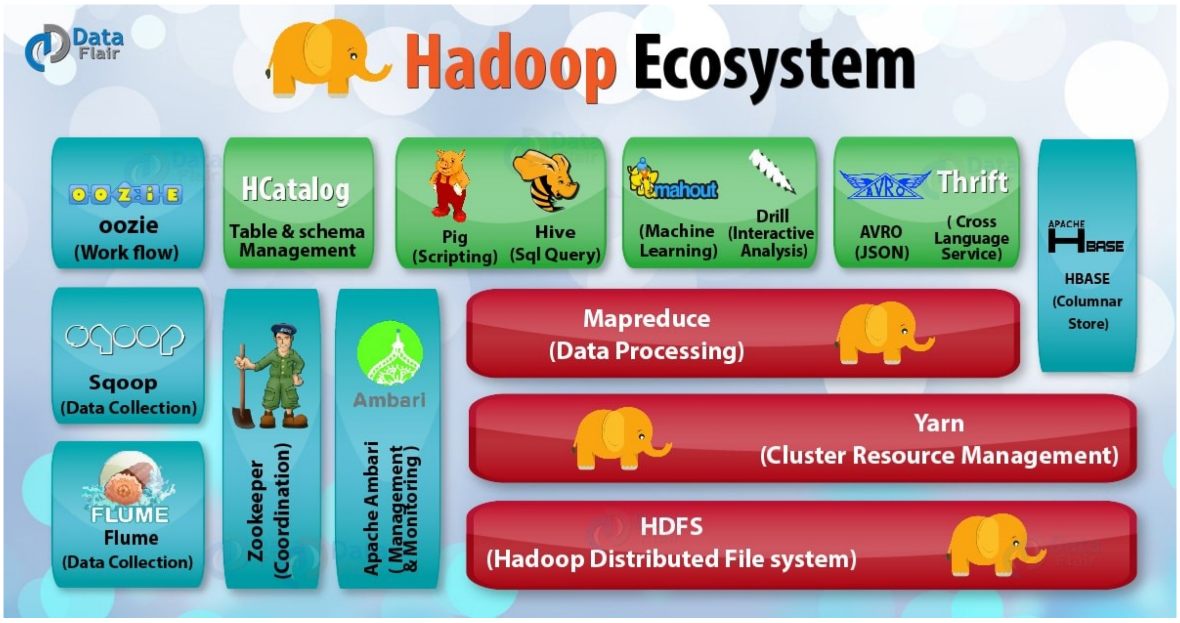


Figure 21 – Apache Hadoop Ecosystem [87]

Going back to the very beginning of "Big Data", among the triggers of all of this are generally considered the search engine providers in the early 2000s, especially Google and Yahoo (AltaVista). The search engine providers were the first group of users faced with Internet scale problems, mainly how to process and store indexes of all of the documents in the Internet universe. Yahoo and Google independently started working on projects to meet this challenge. In 2003, Google released a whitepaper called "The Google File System." [88] Subsequently, in 2004, Google released another whitepaper called "MapReduce: Simplified Data Processing on Large Clusters." [89]. At the same time at Yahoo, Doug Cutting was working on a web indexing project called Nutch (an open source web search engine, and a part of the Lucene project). Google's whitepapers inspired Doug Cutting to take the work he had done to date on the Nutch project and incorporate the storage and processing principles outlined in these whitepapers. By the middle of 2005 Nutch was using both MapReduce and HDFS and the resultant product is what is known today as Hadoop. Hadoop was born in 2006 as an open source project under the Apache Software Foundation under Lucene project [90]. By the beginning of 2008, Hadoop was a top-level project at Apache and was being used by many companies. From that time many other open project originally or later supported by Apache, has hooked up Hadoop and made it a center of what is now so-called Hadoop Ecosystem [91].

Before the most important part of Hadoop Ecosystem will be explained, it's necessary to understand how each part fits together. As Figure 21 shows, there are many projects or components associated with core Hadoop.

Short list of key components of Hadoop Ecosystem is also presented in the Table 12 together with their corresponding area.

Table 12 – Key Components of Hadoop Ecosystem

Area	Project
Distributed Storage	Hadoop HDFS
Resource Management	Hadoop YARN
Processing Framework	Hadoop MapReduce v2, Tez, Hoys
Data Collection/Movement	Sqoop, Flume
Management & Coordination	Ambari, Zookeper, Hue
Workflow Engine, Scheduling	Oozie
Data Serialization	Avro,
Table and Schema Management	HCatalog
Columnar Store	HBase
Scripting	Pig
SQL Query, DWH	Hive
Machine Learning	Mahout, Spark MLlib, Hadoop Submarine
Analytics, Analysis, Processing Engine	Spark, Drill, Impala
Streaming & Messaging	Kafka, Storm
Searching & Indexing	SOLR, Lucene

Ambari – an integrated set of Hadoop administration tools for installing, monitoring, and maintaining a Hadoop cluster. Also included are tools to add or remove slave nodes.

Avro – a framework for the efficient serialization (a kind of transformation) of data into a compact binary format

Flume – a distributed service for collecting and aggregating data from almost any source into a data store such as HDFS or HBase

HBase – a distributed columnar database that uses HDFS for its underlying storage. With HBase, you can store data in extremely large tables with variable column structures

HCatalog – a service for providing a relational view of data stored in Hadoop, including a standard approach for tabular data

Hive – a distributed data warehouse for data that is stored in HDFS; also provides a query language that's based on SQL (HiveQL)

Hue – a Hadoop administration interface with handy GUI tools for browsing files, issuing Hive and Pig queries, and developing Oozie workflows

Mahout – a library of machine learning statistical algorithms such a core algorithms for clustering, classification, and batch-based collaborative filtering that were implemented in MapReduce and can run natively on Hadoop

Oozie – a workflow management tool that can handle the scheduling and chaining together of Hadoop applications

Pig – a platform for the analysis of very large data sets that runs on HDFS and with an infrastructure layer consisting of a compiler that produces sequences of MapReduce programs and a language layer consisting of the query language named Pig Latin

Spark – a fast engine for processing large-scale data. It supports Java, Scala, and Python applications. Because it provides primitives for in-memory cluster computing, it is particularly suited to machine-learning algorithms. It promises performance up to 10 to 100 times faster than MapReduce.

Sqoop – a tool for efficiently moving large amounts of data between relational databases and HDFS or Hive

ZooKeeper – a simple interface to the centralized coordination of services (such as naming, configuration, and synchronization) used by distributed applications

To name all components or projects related to Hadoop or Hadoop Ecosystem would be tough work. Only Apache itself currently list 50 projects in Big Data category and next 24 projects in Database category [92]. Each month new projects are emerging. Therefore, only some of the most important will be presented.

3.1 Apache Hadoop

The Apache Hadoop software library is a framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models. It is designed to scale up from single servers to thousands of machines, each offering local computation and storage. Rather than rely on hardware to deliver high-availability, the library itself is designed to detect and handle failures at the application layer, so delivering a highly-available service on top of a cluster of computers, each of which may be prone to failures [90].

Hadoop is a data storage and processing platform, based upon a central concept of "*Data Locality*" which refers to the processing of data at its side, means bringing the computation to the data, rather than the typical pattern of requesting data from its location and sending them to a remote processing. Once the data reach the volume of "Big Data" it's more efficient to "send" processing to data not vice versa.

Hadoop enables large data sets to be processed locally on the nodes of a cluster using a *shared nothing approach*, where each node can independently process a much smaller subset of the entire data set without needing to communicate with one another.

Hadoop is *schemaless* with respect to its write operations (known as a schema-on-read). This means that it can store and process a different type of data, from unstructured text documents, to semi-structured JSON/XML documents, to well-structured data known from RDBMS. On the contrary, schema-on-write systems, are system, where data are typically strongly typed and a schema is predefined.

Hadoop is designed to apply principle of divide and conquer a large problem into a set of smaller problems and applying the concepts of data locality and shared nothing as previously introduced.

3.1.1 Overview of History and Release Versioning

The very beginning of Hadoop was introduced at the beginning of this chapter. To remind, in 2002/2003 Doug Cutting and Mike Cafarella (known as co-founders of Hadoop) were working in company Yahoo! On web search project called Nutch. In between, in 2003, Google released a whitepaper called “The Google File System.” [88] Subsequently, in 2004, Google released another whitepaper called “MapReduce: Simplified Data Processing on Large Clusters.” [89] Cutting inspired by these paper incorporated these ideas into his Nutch project, and later in January 2006 moved the Nutch project under the Apache Foundation. The initial code that was factored out of Nutch consisted of about 5,000 lines of code for HDFS and about 6,000 lines of code for MapReduce. In the following Table 13 all significant Hadoop events are collected.

Table 13 – Timeline of Hadoop History [93]

Year	Month	Event
2003	October	Google File System paper released
2004	December	MapReduce: Simplified Data Processing on Large Clusters
2006	January	Hadoop subproject created with mailing lists, jira, and wiki
2006	January	Hadoop is born from Nutch 197
2006	February	NDFS+ MapReduce moved out of Apache Nutch to create Hadoop
2006	February	Owen O'Malley's first patch goes into Hadoop
2006	February	Hadoop is named after Cutting's son's yellow plush toy
2006	April	Hadoop 0.1.0 released
2006	April	Hadoop sorts 1.8 TB on 188 nodes in 47.9 hours
2006	May	Yahoo deploys 300 machine Hadoop cluster
2006	October	Yahoo Hadoop cluster reaches 600 machines
2007	April	Yahoo runs two clusters of 1,000 machines
2007	June	Only three companies on "Powered by Hadoop Page"
2007	October	First release of Hadoop that includes HBase
2007	October	Yahoo Labs creates Pig, and donates it to the ASF
2008	January	YARN JIRA opened
2008	January	20 companies on "Powered by Hadoop Page"
2008	February	Yahoo moves its web index onto Hadoop
2008	February	Yahoo! production search index generated by a 10,000-core Hadoop cluster
2008	March	First Hadoop Summit
2008	April	Hadoop world record fastest system to sort a terabyte of data.
2008	May	Hadoop wins TeraByte Sort (World Record sortbenchmark.org)
2008	July	Hadoop wins Terabyte Sort Benchmark
2008	October	Loading 10 TB/day in Yahoo clusters
2008	October	Cloudera, Hadoop distributor is founded
2008	November	Google MapReduce implementation sorted one terabyte in 68 seconds
2009	March	Yahoo runs 17 clusters with 24,000 machines
2009	April	Hadoop sorts a petabyte
2009	May	Yahoo! used Hadoop to sort one terabyte in 62 seconds

Year	Month	Event
2009	June	Second Hadoop Summit
2009	July	Hadoop Core is renamed Hadoop Common
2009	July	MapR, Hadoop distributor founded
2009	July	HDFS now a separate subproject
2009	July	MapReduce now a separate subproject
2010	January	Kerberos support added to Hadoop
2010	May	Apache HBase Graduates
2010	June	Third Hadoop Summit
2010	June	Yahoo 4,000 nodes/70 petabytes
2010	June	Facebook 2,300 clusters/40 petabytes
2010	September	Apache Hive Graduates
2010	September	Apache Pig Graduates
2011	January	Apache Zookeeper Graduates
2011	January	Facebook, LinkedIn, eBay and IBM collectively contribute 200,000 lines of
2011	March	Apache Hadoop takes top prize at Media Guardian Innovation Awards
2011	June	Rob Beardon and Eric Badleschieler spin Hortonworks out of Yahoo.
2011	June	Yahoo has 42K Hadoop nodes and hundreds of petabytes of storage
2011	June	Third Annual Hadoop Summit (1,700 attendees)
2011	October	Debate over which company had contributed more to Hadoop.
2012	January	Hadoop community moves to separate from MapReduce and replace with
2012	June	San Jose Hadoop Summit (2,100 attendees)
2012	November	Apache Hadoop 1.0 Available
2013	March	Hadoop Summit – Amsterdam (500 attendees)
2013	March	YARN deployed in production at Yahoo
2013	June	San Jose Hadoop Summit (2,700 attendees)
2013	October	Apache Hadoop 2.2 Available
2014	February	Apache Hadoop 2.3 Available
2014	February	Apache Spark top Level Apache Project
2014	April	Hadoop summit Amsterdam (750 attendees)
2014	June	Apache Hadoop 2.4 Available
2014	June	San Jose Hadoop Summit (3,200 attendees)
2014	August	Apache Hadoop 2.5 Available
2014	November	Apache Hadoop 2.6 Available
2015	April	Hadoop Summit Europe
2015	June	Apache Hadoop 2.7 Available
2017	March	Apache Hadoop 2.8 Available
2017	November	Apache Hadoop 2.9 Available
2017	December	Apache Hadoop 3.0 Available
2018	April	Apache Hadoop 3.1 Available
2018	September 15	Apache Hadoop 2.8.5 Available
2018	November 19	Apache Hadoop 2.9.2 Available
2019	January 16	Apache Hadoop 3.2 Available
2019	February 6	Apache Hadoop 3.1.2 Available

During last almost two decades Hadoop made a great journey as you can see on its timeline in Table 13. If you focus on last five rows of previous table, there could be seen “deflection” of rule. It’s necessary to mention how Hadoop is releasing its versions. Hadoop Community is actively maintaining two stable release lines in each major release line. Apache Hadoop uses a version format of <**major**>.<**minor**>.<**maintenance**>, where each version component is a numeric value. Versions can also have additional suffixes like ”-alpha1” or ”-beta2”, which denote the API compatibility guarantees and quality of the release. ***Major*** versions are used to introduce *substantial, potentially incompatible, changes*. Examples of this include the replacement of MapReduce 1 with YARN & MapReduce 2 in Hadoop 2, and the required Java runtime version from JDK7 to JDK8 in Hadoop 3. ***Minor*** versions are used to introduce *new compatible features within a major* release line. ***Maintenance*** releases include *bug fixes or low-risk supportability changes*. Hadoop’s versioning scheme has evolved over the years, but from 2.7/2.8 was re-introduced parallel active release lines to Hadoop. With major release 3, four active release lines are maintained [94].

3.1.2 Distribution

Although Hadoop is an open source project there are many commercial vendors who supply commercial distributions, support, management utilities and more. Figure 22 depicts rough estimate of market share. Generally all vendors can be divided into two parts defined by how much the Hadoop is core business for them. First groups of providers could be referred as “pure play” Hadoop vendors. Their business model is base pure on Hadoop. In 2008, the first commercial vendor, Cloudera, was formed by engineers from Google, Yahoo!, and Facebook. Later in 2009, MapR was founded as a company delivering a “Hadoop-derived” software solution implementing a custom adaptation of the Hadoop filesystem (called MapRFS) with Hadoop API compatibility. In 2011, Hortonworks was spun off from Yahoo! as a Hadoop vendor providing a distribution called HDP (Hortonworks Data Platform). Last, fresh news from fall of 2018 happened in this field, when Cloudera and Hortonworks announced the join of both companies and continue under the Cloudera name. Another group of Hadoop vendors are mostly big IT companies like Amazon, IBM, Microsoft and others who offer Hadoop distribution mostly as part of their cloud solutions.

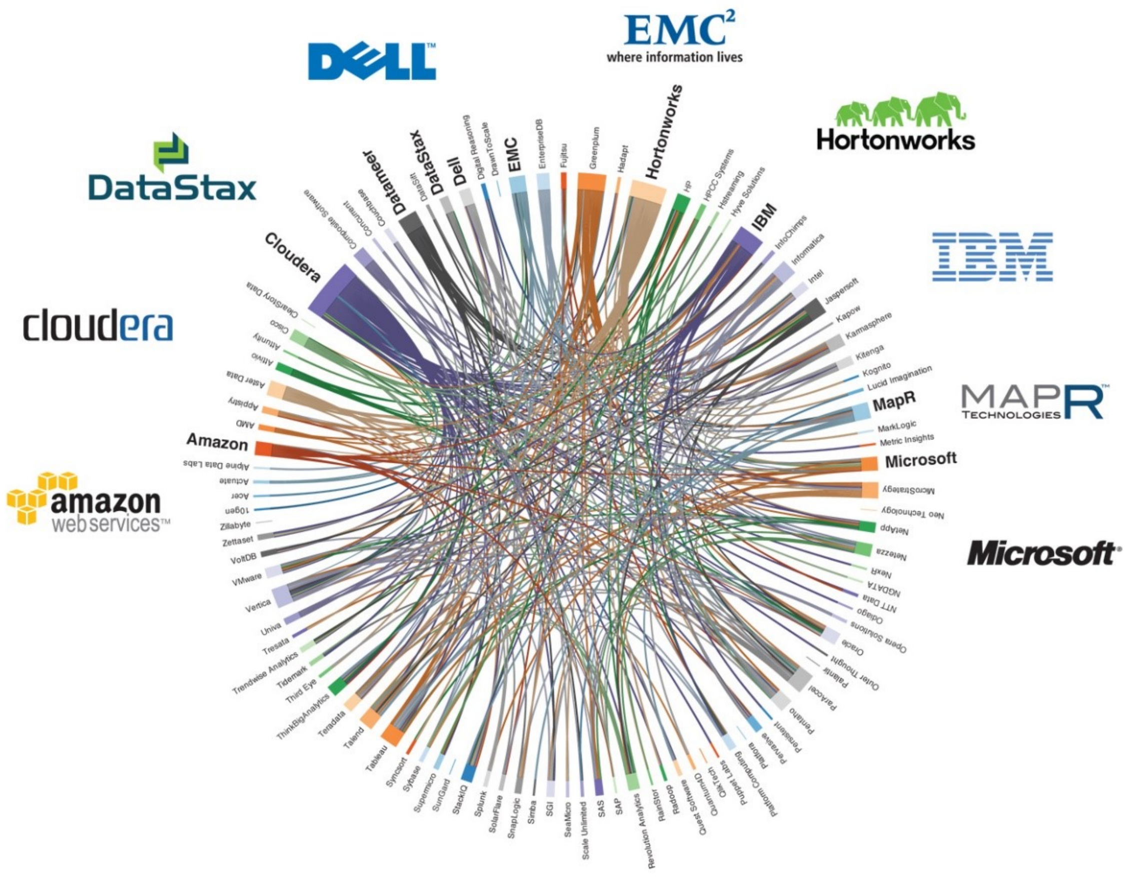


Figure 22 – Hadoop Vendors [95]

3.1.3 Core Components

Hadoop has two core components: Hadoop Distributed File System (HDFS) and YARN (which stands for Yet Another Resource Negotiator). HDFS is Hadoop’s storage subsystem, whereas YARN can be thought of as Hadoop’s process scheduling subsystem (see Figure 23). Next key component is Hadoop Common, the common utilities that support the other Hadoop modules. Historically, from version 1.0 the next component is Hadoop MapReduce, currently in version 2. However, there are new distributed computation models trying to improve MapReduce such Tez [96], that’s why the MapReduce is not considered as key part of Hadoop anymore. Lastly, next two project Hadoop Ozone (an object store for Hadoop) and Hadoop Submarine (a machine learning engine for Hadoop) was taken under umbrella of Hadoop project, but again it is not what makes Hadoop “The Hadoop”.

Each component (HDFS and YARN) is independent of one another and can operate independently in its own cluster, but when they are co-located with one another, the combination of both systems is considered to be a Hadoop cluster.

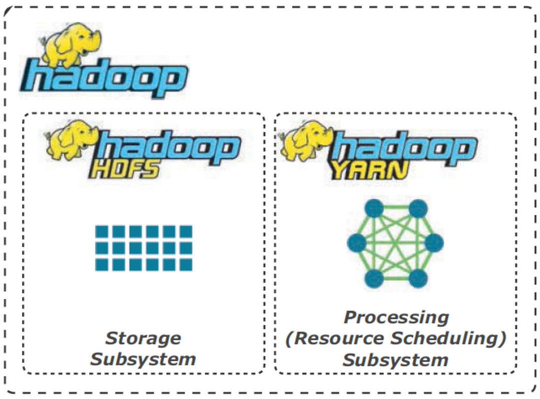


Figure 23 – Key Component of Hadoop

Hadoop Common

Hadoop Common (previously known as Hadoop Core) is one of key modules of Hadoop and refers to the collection of common utilities and libraries that support other Hadoop modules. It is an essential part or module of the Apache Hadoop Framework, along with the Hadoop Distributed File System (HDFS), Hadoop YARN and Hadoop MapReduce. The Hadoop Common package is considered as the base/core of the framework as it provides essential services and basic processes such as abstraction of the underlying operating system and its file system. Hadoop Common also contains the necessary Java Archive (JAR) files and scripts required to start Hadoop. The Hadoop Common package also provides source code and documentation [97].

3.1.4 Deploying Hadoop

Generally, on premise Hadoop deployment can be done in three modes:

- Standalone mode on Single Node Cluster
- Pseudo distributed mode on Single Node Cluster
- Distributed mode on Multi Node Cluster

In this chapter, due to the limited processing power, the Pseudo distributed mode will be presented.

The Hadoop project, although cross-platform in principle, was originally targeted at Linux. However, now is also available on Windows platform, Hadoop will be presented on Linux with latest available versions in time of writing this thesis (April 2019) as other product from Hadoop Ecosystem has known issues with non-Linux operating system. These versions of installed software are used:

- Ubuntu Server – 18.04.2 LTS Bionic
- Java 8 – JDK 1.8.212
- Apache Hadoop – 3.1.2

Conventions

These **styles of code snippets** are used in following paragraphs:

```
# linux commands and parameters or configuration snippets
```

```
roman@hadoop-00:~$ command and parameters
with console output snippets
```

```
public class JavaOrPythonOrScalaCode {
  public static void main(String[] args) throws Exception {
```

Apache Hadoop Installation Steps

Installation of Hadoop system consists of several steps. Here is a quick summary, and further each step is explained with commands/configuration and typical outputs of console.

1) Java Installation

2) SSH Configuration

3) Hadoop Installation

4) Setup Hadoop Configuration Files

5) Format Namenode

6) Start Hadoop Cluster

7) Test Hadoop Cluster

Now it's time to start with first prerequisite, and it is Java installation.

3.1.4.1 Java Installation

It's expected that clean installation of Ubuntu Server has been performed. Java is the primary requirement for running Hadoop on any system, so first is necessary to install Java on host server.

1) **Install Java** using apt command:

```
# sudo apt install openjdk-8-jdk
```

```
roman@hadoop-00:~$ sudo apt install openjdk-8-jdk
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  adwaita-icon-theme at-spi2-core ca-certificates-java ...
...
ovide /usr/bin/jconsole (jconsole) in auto mode
Processing triggers for libc-bin (2.27-3ubuntu1) ...
Processing triggers for ureadahead (0.100.0-21) ...
Processing triggers for systemd (237-3ubuntu10.21) ...
Processing triggers for libgdk-pixbuf2.0-0:amd64 (2.36.11-2) ...
roman@hadoop-00:~$
```

2) After installation, to **verify the java** has been successfully configured, run the following commands:

```
# update-alternatives --display java
# update-alternatives --display javac
```

```
roman@hadoop-00:~$ update-alternatives --display java
java - auto mode
  link best version is /usr/lib/jvm/java-8-openjdk-amd64/jre/bin/java
  link currently points to /usr/lib/jvm/java-8-openjdk-amd64/jre/bin/java
  link java is /usr/bin/java
  slave java.1.gz is /usr/share/man/man1/java.1.gz
/usr/lib/jvm/java-8-openjdk-amd64/jre/bin/java - priority 1081
  slave java.1.gz: /usr/lib/jvm/java-8-openjdk-amd64/jre/man/man1/java.1.gz
roman@hadoop-00:~$ update-alternatives --display javac
javac - auto mode
  link best version is /usr/lib/jvm/java-8-openjdk-amd64/bin/javac
  link currently points to /usr/lib/jvm/java-8-openjdk-amd64/bin/javac
  link javac is /usr/bin/javac
  slave javac.1.gz is /usr/share/man/man1/javac.1.gz
/usr/lib/jvm/java-8-openjdk-amd64/bin/javac - priority 1081
  slave javac.1.gz: /usr/lib/jvm/java-8-openjdk-amd64/man/man1/javac.1.gz
roman@hadoop-00:~$
```

3.1.4.2 SSH Configuration

SSH (Secure SHELL) is an open source and most trusted network protocol that is used to login into remote servers for execution of commands and programs. It is also used to transfer files from one computer to another computer over the network using secure copy (SCP) Protocol. SSH is required in Hadoop to manage its nodes, i.e. remote machines and local machine, if you want to use Hadoop on it. Using Password-less login with SSH keys will increase the trust between two Linux servers for easy file synchronization or transfer.

1) Installation

If SSH-Server and SSH-Client wasn't installed in time of operating system installation, it's necessary to install now:

```
# sudo apt-get install openssh-server openssh-client
```

2) Generate Key Pairs

Next step is to generate Public and Private Key Pairs with the following command.

```
# ssh-keygen -t rsa -P '' -f ~/.ssh/id_rsa
```

```
roman@hadoop-00:~$ ssh-keygen -t rsa -P '' -f ~/.ssh/id_rsa
Generating public/private rsa key pair.
Created directory '/home/roman/.ssh'.
Your identification has been saved in /home/roman/.ssh/id_rsa.
Your public key has been saved in /home/roman/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:ejJz4npkYQBtMm4BekNIzzdf25MUd4AclCF9o54mKEA roman@hadoop-00
The key's randomart image is:
+---[RSA 2048]----+
|o+oo     .+==o.. |
|o.*E+     .=oo.  |
|..+B +   . .o .  |
| .oo. = . +..    |
| .  .. oS..+.    |
|     .oo . +.    |
|     oB o o      |
|     ..B         |
|    .o.          |
+----[SHA256]-----+
roman@hadoop-00:~$
```

3) Configure passwordless ssh

```
# cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
```

```
roman@hadoop-00:~$ cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
```

4) Change the permission of file that contains the key

```
# chmod 0600 ~/.ssh/authorized_keys
```

```
roman@hadoop-00:~$ chmod 0600 ~/.ssh/authorized_keys
```

5) Verification of SSH

Final step is to verify key based login. Below command should not ask for the password but the first time it will prompt for adding RSA to the list of known hosts.

```
# ssh localhost
```

```
roman@hadoop-00:~$ ssh localhost
The authenticity of host 'localhost (::1)' can't be established.
ECDSA key fingerprint is SHA256:XHFMf57ZqtXaEyPc5YOFnJKiuFVENxZwo0y46kDXkpE.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'localhost' (ECDSA) to the list of known hosts.
Welcome to Ubuntu 18.04.2 LTS (GNU/Linux 4.15.0-47-generic x86_64)

  System load:  0.0               Processes:             91
  Usage of /:   44.3% of 9.78GB   Users logged in:       1
  Memory usage: 10%               IP address for enp0s3: 10.0.2.15
  Swap usage:   0%

0 packages can be updated.
0 updates are security updates.

roman@hadoop-00:~$ exit
logout
Connection to localhost closed.
roman@hadoop-00:~$ ssh localhost
Welcome to Ubuntu 18.04.2 LTS (GNU/Linux 4.15.0-47-generic x86_64)

  System load:  0.02              Processes:             91
  Usage of /:   44.3% of 9.78GB   Users logged in:       1
  Memory usage: 10%               IP address for enp0s3: 10.0.2.15
  Swap usage:   0%

0 packages can be updated.
0 updates are security updates.

roman@hadoop-00:~$
```

3.1.4.3 Hadoop Installation

However Hadoop can run under root or other user with some root privileges, it’s strictly recommended to create special user just for Hadoop.

```
# sudo groupadd hadoopgrp
# sudo adduser -ingroup hadoopgrp hadoop
```

In next step, **download** installation file (under hadoop user):

```
#  wget http://hadoop.bistaff.eu/hadoop-3.1.2.tar.gz
```

```
hadoop@hadoop-00:~$ wget http://hadoop.bistaff.eu/hadoop-3.1.2.tar.gz
--2019-04-23 18:30:43--  http://hadoop.bistaff.eu/hadoop-3.1.2.tar.gz
Resolving hadoop.bistaff.eu (hadoop.bistaff.eu)... 217.11.249.141,
2001:1528:240::1f
Connecting to hadoop.bistaff.eu (hadoop.bistaff.eu)|217.11.249.141|:80...
connected.
HTTP request sent, awaiting response... 200 OK
Length: 332433589 (317M) [application/x-gzip]
Saving to: ‘hadoop-3.1.2.tar.gz’

hadoop-3.1.2.tar.gz 100%[===================>] 317.03M  40.9MB/s    in 7.8s

2019-04-23 18:30:51 (40.7 MB/s) - ‘hadoop-3.1.2.tar.gz’ saved
[332433589/332433589]

hadoop@hadoop-00:~$
```

Extract the installation file.

```
#  tar -xzvf hadoop-3.1.2.tar.gz
```

Setting up the environment variables. Add these configuration lines into *.bashrc*.

```
export HADOOP_HOME=/home/hadoop/hadoop-3.1.2
export HADOOP_INSTALL=$HADOOP_HOME
export HADOOP_MAPRED_HOME=$HADOOP_HOME
export HADOOP_COMMON_HOME=$HADOOP_HOME
export HADOOP_HDFS_HOME=$HADOOP_HOME
export YARN_HOME=$HADOOP_HOME
export HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_HOME/lib/native
export PATH=$PATH:$HADOOP_HOME/sbin:$HADOOP_HOME/bin
export HADOOP_OPTS="-Djava.library.path=$HADOOP_HOME/lib/native"
```

And source the *.bashrc* in current login session for immediate applying changes previously done.

```
#  source ~/.bashrc
```

Edit the *$HADOOP_HOME/etc/hadoop/**hadoop-env.sh*** file inside the Hadoop installation directory

```
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
export HADOOP_CONF_DIR=${HADOOP_CONF_DIR:-"/home/hadoop/hadoop-
3.1.2/etc/hadoop"}
```

3.1.4.4 Setup Hadoop Configuration Files

Hadoop has many of configuration files, which need to configure as per requirements of your Hadoop infrastructure. Start with the configuration with basic Hadoop single node cluster setup.

core-site.xml

Edit *$HADOOP_HOME/etc/hadoop/core-site.xml*

```
<configuration>
  <property>
    <name>fs.defaultFS</name>
    <value>hdfs://localhost:9000</value>
  </property>
  <property>
    <name>hadoop.tmp.dir</name>
    <value>/home/hadoop/hadooptmpdata</value>
  </property>
</configuration>
```

In addition, create the directory under *hadoop* home folder.

```
# mkdir hadooptmpdata
```

hdfs-site.xml

Edit *$HADOOP_HOME/etc/hadoop/hdfs-site.xml*

```
<configuration>
  <property>
    <name>dfs.replication</name>
    <value>1</value>
    <name>dfs.name.dir</name>
    <value>file:///home/hadoop/hdfs/namenode</value>
    <name>dfs.data.dir</name>
    <value>file:///home/hadoop/hdfs/datanode</value>
  </property>
</configuration>
```

In addition, create the directory under *hadoop* home folder.

```
# mkdir -p hdfs/namenode
# mkdir -p hdfs/datanode
```

mapred-site.xml

Edit *$HADOOP_HOME/etc/hadoop/**mapred-site.xml***

```
<configuration>
  <property>
    <name>mapreduce.framework.name</name>
    <value>yarn</value>
  </property>

  <!-- add after pi test -->
  <property>
    <name>yarn.app.mapreduce.am.env</name>
    <value>HADOOP_MAPRED_HOME=${HADOOP_HOME}</value>
  </property>
  <property>
    <name>mapreduce.map.env</name>
    <value>HADOOP_MAPRED_HOME=${HADOOP_HOME}</value>
  </property>
  <property>
    <name>mapreduce.reduce.env</name>
    <value>HADOOP_MAPRED_HOME=${HADOOP_HOME}</value>
  </property>
</configuration>
```

yarn-site.xml

Edit *$HADOOP_HOME/etc/hadoop/**yarn-site.xml***

```
<configuration>
  <property>
    <name>yarn.nodemanager.aux-services</name>
    <value>mapreduce_shuffle</value>
  </property>
  <property>
    <name>yarn.nodemanager.vmem-check-enabled</name>
    <value>false</value>
  </property>
</configuration>
```

3.1.4.5 Format Namenode

Namenode is the node in the Hadoop Distributed File System which keeps track of all the data stored in the Datanode. Namenode has metadata related to the data stored on the Datanodes and has information related to the location of the data stored. So, when you run the *hadoop namenode -format* command, all these information is deleted from the namenode which means that the system does not know where the data are stored hence losing all data stored in the Hadoop File System.

The purpose of formatting the HDFS file system is to create the necessary directory and file structure for further storing data used by MapReduce, especially metadata, temporary data, source data, and result data. The formatting of the HDFS file system itself does not disturb the operating system's file system, because it is built on top of it - it is just a logical file system at the application level, not at the operating system or hardware level.

Format the namenode before using it for the first time!

```
# hdfs namenode -format
```

```
hadoop@hadoop-00:~/hadoop-3.1.2$ hdfs namenode -format
WARNING: /home/hadoop/hadoop-3.1.2/logs does not exist. Creating.
2019-04-23 19:29:06,500 INFO namenode.NameNode: STARTUP_MSG:
/************************************************************
STARTUP_MSG: Starting NameNode
STARTUP_MSG:   host = hadoop-00/10.0.2.15
STARTUP_MSG:   args = [-format]
STARTUP_MSG:   version = 3.1.2
STARTUP_MSG:   classpath = /home/hadoop/hadoop-
3.1.2/etc/hadoop:/home/hadoop/hadoop-3.1.2...
STARTUP_MSG:   build = https://github.com/apache/hadoop.git -r
1019dde65bcf12e05ef48ac71e...
STARTUP_MSG:   java = 1.8.0_191
************************************************************/
2019-04-23 19:29:06,521 INFO namenode.NameNode: registered UNIX signal
handlers for [TERM, HUP, INT]
2019-04-23 19:29:06,733 INFO namenode.NameNode: createNameNode [-format]
Formatting using clusterid: CID-0040c95e-4e8b-4570-9cd5-b455f2c5c5bd
2019-04-23 19:29:07,776 INFO namenode.FSEditLog: Edit logging is async:true
2019-04-23 19:29:07,817 INFO namenode.FSNamesystem: KeyProvider: null
2019-04-23 19:29:07,819 INFO namenode.FSNamesystem: fsLock is fair: true
2019-04-23 19:29:07,821 INFO namenode.FSNamesystem: Detailed lock hold time
metrics enabled: false
2019-04-23 19:29:07,828 INFO namenode.FSNamesystem: fsOwner             =
hadoop (auth:SIMPLE)
2019-04-23 19:29:07,829 INFO namenode.FSNamesystem: supergroup          =
supergroup
2019-04-23 19:29:07,829 INFO namenode.FSNamesystem: isPermissionEnabled = true
...
2019-04-23 19:29:07,982 INFO blockmanagement.BlockManagerSafeMode:
dfs.namenode.safemode.threshold-pct = 0.9990000128746033
2019-04-23 19:29:07,982 INFO blockmanagement.BlockManagerSafeMode:
dfs.namenode.safemode.min.datanodes = 0
2019-04-23 19:29:07,982 INFO blockmanagement.BlockManagerSafeMode:
dfs.namenode.safemode.extension = 30000
2019-04-23 19:29:07,983 INFO blockmanagement.BlockManager: defaultReplication
= 3
2019-04-23 19:29:07,983 INFO blockmanagement.BlockManager: maxReplication
= 512
2019-04-23 19:29:07,984 INFO blockmanagement.BlockManager: minReplication
= 1
2019-04-23 19:29:07,984 INFO blockmanagement.BlockManager:
maxReplicationStreams      = 2
2019-04-23 19:29:07,984 INFO blockmanagement.BlockManager:
redundancyRecheckInterval  = 3000ms
```

```
2019-04-23 19:29:07,985 INFO blockmanagement.BlockManager: encryptDataTransfer
= false
2019-04-23 19:29:07,985 INFO blockmanagement.BlockManager: maxNumBlocksToLog
= 1000
...
2019-04-23 19:29:08,401 INFO namenode.NameNode: SHUTDOWN_MSG:
/************************************************************
SHUTDOWN_MSG: Shutting down NameNode at hadoop-00/10.0.2.15
************************************************************/
hadoop@hadoop-00:~/hadoop-3.1.2$
```

3.1.4.6 Start Hadoop Cluster

Once the Namenode has been formatted then start the HDFS using the `start-dfs.sh` script.

```
# start-dfs.sh
```

In case of error "Permission denied (publickey, password)." regenerate ssh keys for password-less authentication.

```
# ssh-keygen -t rsa -P '' -f ~/.ssh/id_rsa
# cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
```

Otherwise you should get output similar to this one:

```
hadoop@hadoop-00:~/hadoop-3.1.2$ start-dfs.sh
Starting namenodes on [localhost]
Starting datanodes
Starting secondary namenodes [hadoop-00]
hadoop@hadoop-00:~/hadoop-3.1.2$
```

You can check successful start of HDFS with `jps` command and output should contains *NameNode*, *DataNode* and *SecondaryNameNode*.

```
# jps
```

```
hadoop@hadoop-00:~$ jps
2807 Jps
2258 NameNode
2426 DataNode
2683 SecondaryNameNode
hadoop@hadoop-00:~$
```

Similarly start YARN.

```
# start-yarn.sh
```

```
hadoop@hadoop-00:~/hadoop-3.1.2$ start-yarn.sh
Starting resourcemanager
Starting nodemanagers
hadoop@hadoop-00:~/hadoop-3.1.2$
```

Output of check if YARN was successfully started should contains: *NodeManager* and *ResourceManager*.

3.1.4.7 Test Hadoop Cluster

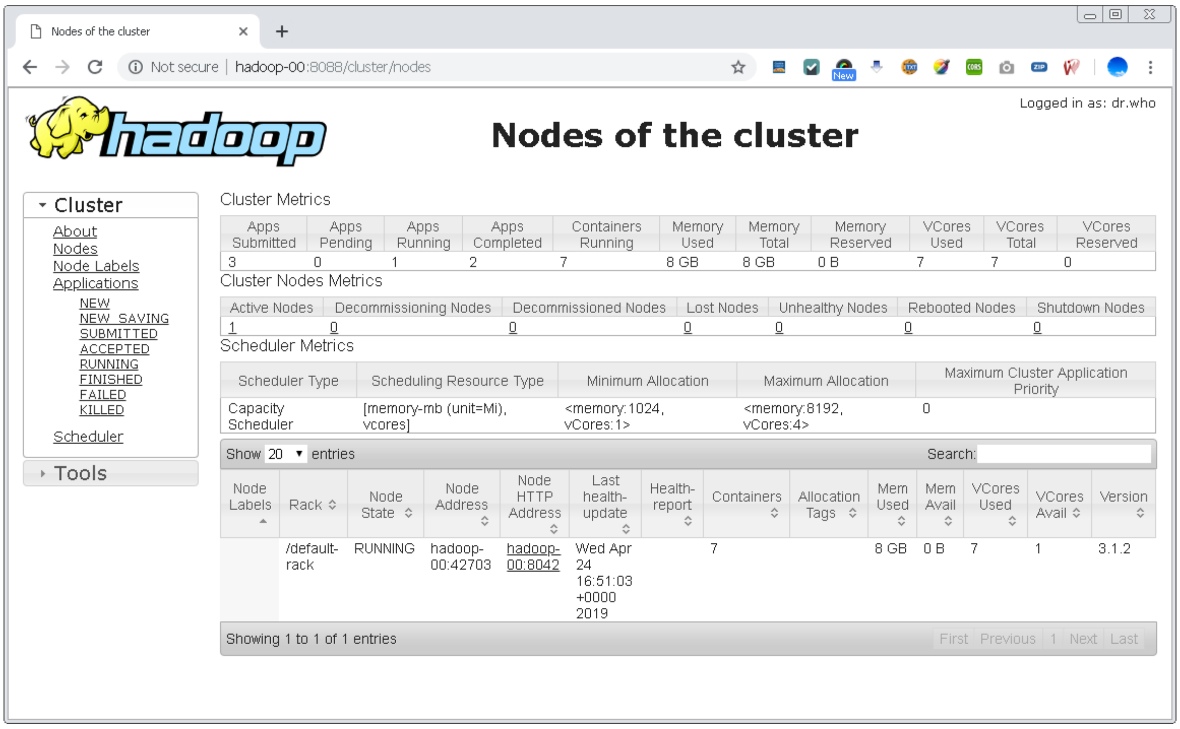
Now it’s possible to test first MapReduce job and confirm that Hadoop is working properly. Run this command included in with the Hadoop release.

```
#  bin/hadoop jar share/hadoop/mapreduce/hadoop-mapreduce-examples-3.1.2.jar
pi 8 10000
```

```
hadoop@hadoop-00:~/hadoop-3.1.2$ bin/hadoop jar share/hadoop/mapreduce/hadoop-
mapreduce-examples-3.1.2.jar pi 24 1000
Number of Maps  = 24
Samples per Map = 1000
Wrote input for Map #0
Wrote input for Map #1
Wrote input for Map #2
...[skipped some lines]...
Wrote input for Map #22
Wrote input for Map #23
Starting Job
2019-04-24 16:23:03,541 INFO client.RMProxy: Connecting to ResourceManager at
/0.0.0.0:8032
2019-04-24 16:23:04,013 INFO mapreduce.JobResourceUploader: Disabling Erasure
Coding for path: /tmp/hadoop-
yarn/staging/hadoop/.staging/job_1556122862873_0002
2019-04-24 16:23:04,289 INFO input.FileInputFormat: Total input files to
process : 24
2019-04-24 16:23:04,437 INFO mapreduce.JobSubmitter: number of splits:24
2019-04-24 16:23:04,794 INFO mapreduce.JobSubmitter: Submitting tokens for
job: job_1556122862873_0002
2019-04-24 16:23:04,796 INFO mapreduce.JobSubmitter: Executing with tokens: []
2019-04-24 16:23:05,191 INFO conf.Configuration: resource-types.xml not found
2019-04-24 16:23:05,193 INFO resource.ResourceUtils: Unable to find 'resource-
types.xml'.
2019-04-24 16:23:05,300 INFO impl.YarnClientImpl: Submitted application
application_1556122862873_0002
2019-04-24 16:23:05,382 INFO mapreduce.Job: The url to track the job:
http://hadoop-00:8088/proxy/application_1556122862873_0002/
2019-04-24 16:23:05,385 INFO mapreduce.Job: Running job:
job_1556122862873_0002
2019-04-24 16:23:16,776 INFO mapreduce.Job: Job job_1556122862873_0002 running
in uber mode : false
2019-04-24 16:23:16,789 INFO mapreduce.Job:  map 0% reduce 0%
2019-04-24 16:23:51,182 INFO mapreduce.Job:  map 4% reduce 0%
2019-04-24 16:23:52,270 INFO mapreduce.Job:  map 25% reduce 0%
```

```
2019-04-24 16:24:25,508 INFO mapreduce.Job:  map 46% reduce 0%
2019-04-24 16:24:34,562 INFO mapreduce.Job:  map 46% reduce 15%
2019-04-24 16:24:53,705 INFO mapreduce.Job:  map 63% reduce 15%
2019-04-24 16:24:54,713 INFO mapreduce.Job:  map 67% reduce 15%
2019-04-24 16:24:59,738 INFO mapreduce.Job:  map 67% reduce 22%
2019-04-24 16:25:22,879 INFO mapreduce.Job:  map 88% reduce 22%
2019-04-24 16:25:23,893 INFO mapreduce.Job:  map 88% reduce 29%
2019-04-24 16:25:39,986 INFO mapreduce.Job:  map 100% reduce 29%
2019-04-24 16:25:40,990 INFO mapreduce.Job:  map 100% reduce 100%
2019-04-24 16:25:42,030 INFO mapreduce.Job: Job job_1556122862873_0002
completed successfully
2019-04-24 16:25:42,154 INFO mapreduce.Job: Counters: 53
        File System Counters
                FILE: Number of bytes read=534
                FILE: Number of bytes written=5410465
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=6374
                HDFS: Number of bytes written=215
                HDFS: Number of read operations=101
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=3
        Job Counters
                Launched map tasks=24
                Launched reduce tasks=1
                Data-local map tasks=24
                Total time spent by all maps in occupied slots (ms)=673270
                Total time spent by all reduces in occupied slots (ms)=106628
                Total time spent by all map tasks (ms)=673270
                Total time spent by all reduce tasks (ms)=106628
                Total vcore-milliseconds taken by all map tasks=673270
                Total vcore-milliseconds taken by all reduce tasks=106628
                Total megabyte-milliseconds taken by all map tasks=689428480
                Total megabyte-milliseconds taken by all reduce
tasks=109187072
        Map-Reduce Framework
                Map input records=24
                Map output records=48
                Map output bytes=432
                Map output materialized bytes=672
                Input split bytes=3542
                Combine input records=0
                Combine output records=0
                Reduce input groups=2
                Reduce shuffle bytes=672
                Reduce input records=48
                Reduce output records=0
                Spilled Records=96
                Shuffled Maps =24
                Failed Shuffles=0
                Merged Map outputs=24
                GC time elapsed (ms)=10494
                CPU time spent (ms)=15940
                Physical memory (bytes) snapshot=5531590656
                Virtual memory (bytes) snapshot=64717635584
                Total committed heap usage (bytes)=4091183104
                Peak Map Physical memory (bytes)=227823616
                Peak Map Virtual memory (bytes)=2588368896
                Peak Reduce Physical memory (bytes)=120942592
                Peak Reduce Virtual memory (bytes)=2596782080
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=2832
        File Output Format Counters
                Bytes Written=97
Job Finished in 158.78 seconds
Estimated value of Pi is 3.14216666666666666667
hadoop@hadoop-00:~/hadoop-3.1.2$
```

The YARN Resource Manager (RM) web interface will display all running jobs on current Hadoop Cluster. All MapReduce jobs can be checked via browser at address *`http://hadoop-00:8088`* (see Figure 24).


Nodes of the cluster

Not secure | hadoop-00:8088/cluster/nodes

Logged in as: dr.who

Nodes of the cluster

Cluster
About
Nodes
Node Labels
Applications
NEW
NEW_SAVING
SUBMITTED
ACCEPTED
RUNNING
FINISHED
FAILED
KILLED
Scheduler
Tools

Cluster Metrics

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved	VCores Used	VCores Total	VCores Reserved
3	0	1	2	7	8 GB	8 GB	0 B	7	7	0

Cluster Nodes Metrics

Active Nodes	Decommissioning Nodes	Decommissioned Nodes	Lost Nodes	Unhealthy Nodes	Rebooted Nodes	Shutdown Nodes
1	0	0	0	0	0	0

Scheduler Metrics

Scheduler Type	Scheduling Resource Type	Minimum Allocation	Maximum Allocation	Maximum Cluster Application Priority
Capacity Scheduler	[memory-mb (unit=Mi), vcores]	<memory:1024, vCores:1>	<memory:8192, vCores:4>	0

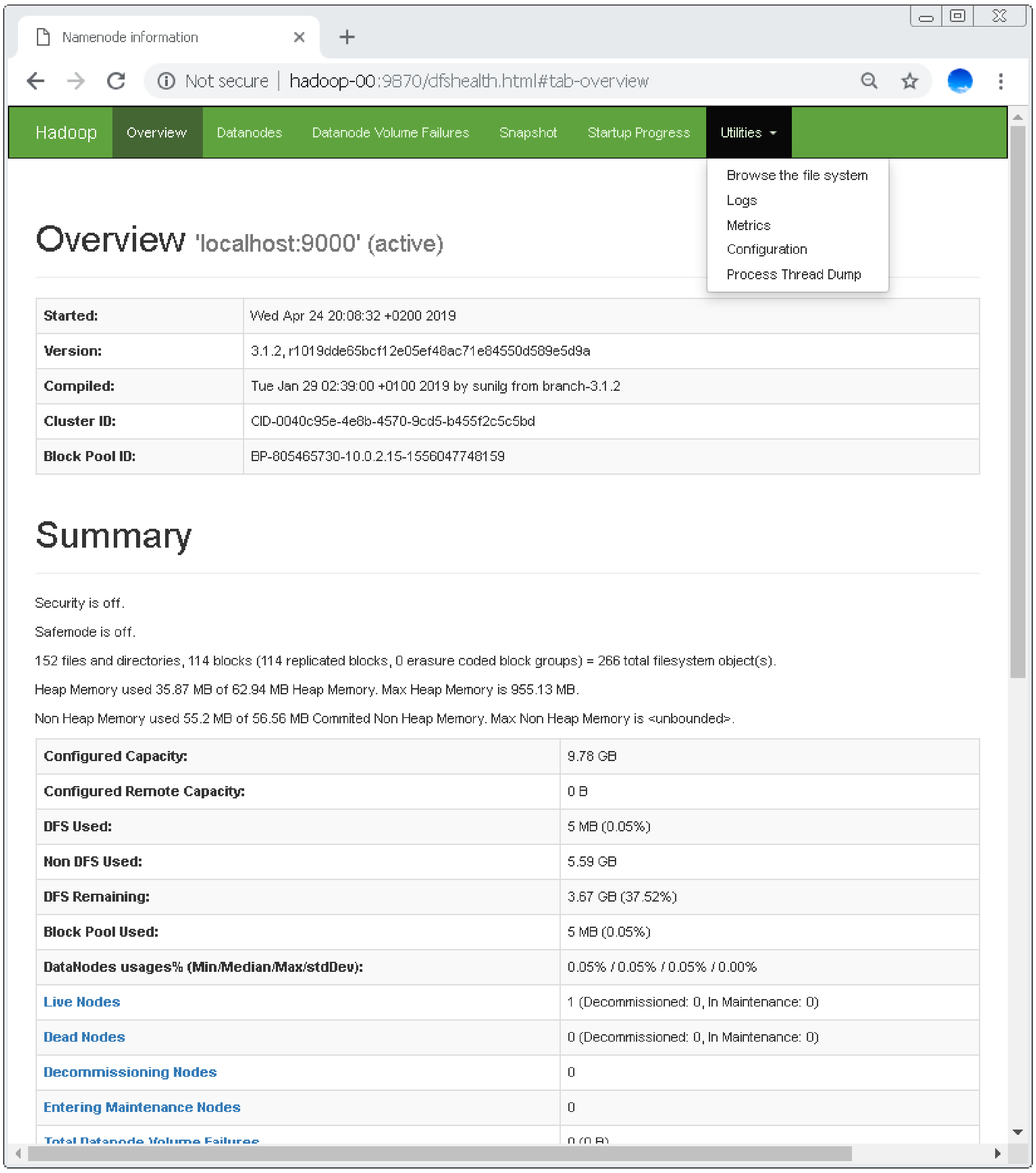
Show 20 entries Search:

Node Labels	Rack	Node State	Node Address	Node HTTP Address	Last health-update	Health-report	Containers	Allocation Tags	Mem Used	Mem Avail	VCores Used	VCores Avail	Version
	/default-rack	RUNNING	hadoop-00:42703	hadoop-00:8042	Wed Apr 24 16:51:03 +0000 2019		7		8 GB	0 B	7	1	3.1.2

Showing 1 to 1 of 1 entries

Figure 24 – Hadoop Cluster Management Console

Another great source of information about Namenodes/Datanodes can be found in Web UI for Namenode at port 9870 (see Figure 25).



Overview 'localhost:9000' (active)

Started:	Wed Apr 24 20:08:32 +0200 2019
Version:	3.1.2, r1019dde65bcf12e05ef48ac71e84550d589e5d9a
Compiled:	Tue Jan 29 02:39:00 +0100 2019 by sunilg from branch-3.1.2
Cluster ID:	CID-0040c95e-4e8b-4570-9cd5-b455f2c5c5bd
Block Pool ID:	BP-805465730-10.0.2.15-1556047748159

Summary

Security is off.

Safemode is off.

152 files and directories, 114 blocks (114 replicated blocks, 0 erasure coded block groups) = 266 total filesystem object(s).

Heap Memory used 35.87 MB of 62.94 MB Heap Memory. Max Heap Memory is 955.13 MB.

Non Heap Memory used 55.2 MB of 56.56 MB Commited Non Heap Memory. Max Non Heap Memory is <unbounded>.

Configured Capacity:	9.78 GB
Configured Remote Capacity:	0 B
DFS Used:	5 MB (0.05%)
Non DFS Used:	5.59 GB
DFS Remaining:	3.67 GB (37.52%)
Block Pool Used:	5 MB (0.05%)
DataNodes usages% (Min/Median/Max/stdDev):	0.05% / 0.05% / 0.05% / 0.00%
Live Nodes	1 (Decommissioned: 0, In Maintenance: 0)
Dead Nodes	0 (Decommissioned: 0, In Maintenance: 0)
Decommissioning Nodes	0
Entering Maintenance Nodes	0
Total Datanode Volume Failures	0 (0 B)

Figure 25 – Datanodes UI

3.1.5 HDFS

The Hadoop Distributed Filesystem (HDFS) is Hadoop's storage platform. Although Hadoop can interact with many different filesystems (Amazon S3, Azure Blob Storage, Azure Data Lake Storage, OpenStack Swift), HDFS is Hadoop's primary input data source and target for data processing operations. Hadoop was originally developed as a platform to support the requirements of search engine providers such as Yahoo!. HDFS was inspired by the GoogleFS.

HDFS is a virtual filesystem, meaning that a client can see it as if it is one system, but the underlying data are located in multiple different locations. HDFS is deployed on top of native filesystems of each particular operating system. HDFS is immutable, it means the inability to update data after it is committed to the filesystem. HDFS is often also referred to as a WORM (write once, read many) filesystem. Given that this filesystem is designed to handle large amounts of data, emphasis has been placed on their fast availability of data for reading than as an interactive data store for users. HDFS does not support POSIX. HDFS puts emphasis on fast file reading and does not allow for frequent modification. Every modification of a file requires re-replication of its copies and thus the load on the network, which is a costly process. However, the latest versions support appending content at the end of a file and overwriting file with a null.

HDFS has been designed to effectively store large amounts of data - gigabytes to terabytes. In theory, it supports storing dozens of millions of files in one HDFS instance. But this is related to the “problem of small files”. A small file is one which is significantly smaller than the HDFS block size (default 64MB). If you’re storing small files, then you probably have lots of them (otherwise you wouldn’t turn to Hadoop), and the problem is that HDFS can’t handle lots of files. Every file, directory and block in HDFS is represented as an object in the namenode’s memory, each of which occupies 150 bytes, as a rule of thumb. So 10 million files, each using a block, would use about 3 gigabytes of memory. Scaling up much beyond this level is a problem with current hardware. Certainly a billion files is not feasible. Furthermore, HDFS is not geared up to efficiently accessing small files: it is primarily designed for streaming access of large files. Reading through small files normally causes lots of seeks and lots of hopping from datanode to datanode to retrieve each small file, all of which is an inefficient data access pattern [98].

Files in HDFS consist of blocks. HDFS splits huge files into small chunks known as data blocks. Default size of HDFS blocks is 128MB, although this can be configured as needed. When the file is loaded into the system, is split into these blocks. All blocks of the file are the same size except the last block, which can be either the same size or smaller (for graphical representation how data are stored into blocks, please see Figure 26).

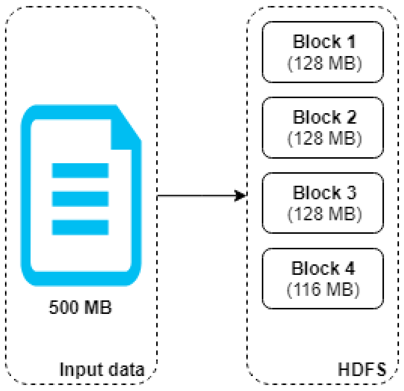


Figure 26 – Data distribution
in HDFS Blocks

If a cluster contains more than one node, blocks are distributed among slave nodes in the cluster in time of loading data into HDFS as shows Figure 27.

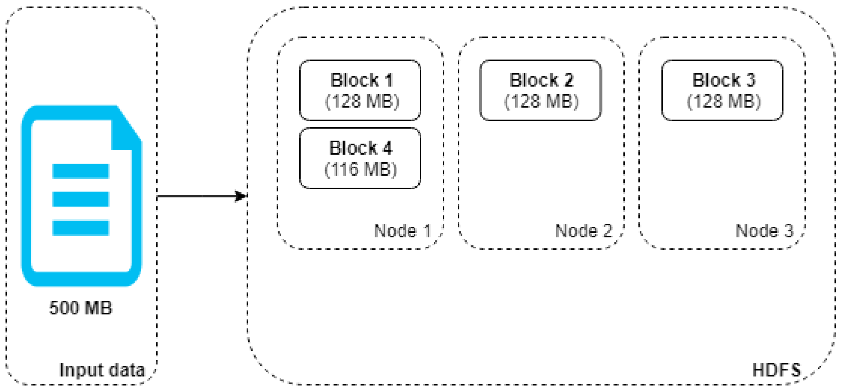


Figure 27 – Data distribution in HDFS across Slave Nodes

3.1.5.1 HDFS Architecture

HDFS is architecturally composed of two demons as is shown in Figure 28. Typically, one master (Namenode) and a few slaves (Datanodes). It is therefore a master / slave architecture. Namenode - the master of the cluster manages, among other things, file metadata (their location, replication factor) and controls access rights to them. Datanodes - slave servers contain data stored in HDFS itself. From a technical point of view, each file is divided into one or more blocks (one 128MB by default) and these blocks are stored on the Datanodes. Namenode performs file operations such as opening, closing, renaming files and folders, and specifying where and on which Datanodes the blocks will be stored. Datanodes are responsible for executing Namenode's requests (from which are requested by the client), such as read and write. Datanodes are also responsible for block creation, deletion, and replication operations.

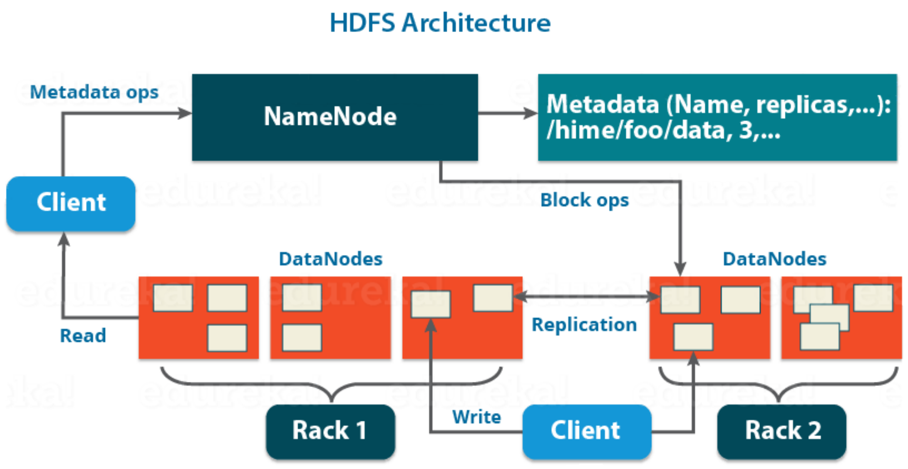


Figure 28 – HDFS Architecture

3.1.5.2 Replication

Each file that is uploaded to HDFS is replicated immediately after uploading with the set replication factor. This variable can be set for the whole cluster in `dfs.replication` parameter, which is by default 3 (see Figure 29). After uploading the file is divided into blocks according to the parameter `dfs.blocksize`, which is 128 MB by default. With this replication factor, the file blocks will be replicated to the three Datanodes. Namenode takes care of where the replica will be stored. It keeps track of the number of available and functional Datanodes using a periodic (`dfs.heartbeat.interval`) signal sending, called "Heart Beat". It also receives data from Datanodes on files stored on them, called "Block Report". If the Block Report shows that some files do not meet the replication factor, then they are "under replicated" files over which replication can be enforced manually.

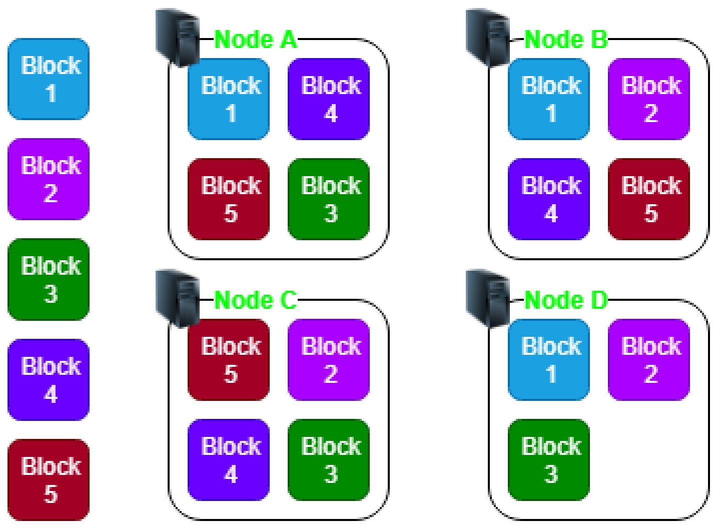


Figure 29 – HDFS Replication

3.1.5.3 Data Read and Write Operations in HDFS

As already mentioned earlier, HDFS is the storage layer of Hadoop. HDFS works in master-slave fashion, Namenode is the master daemon which runs on the master node, and Datanode is the slave daemon which runs on the slave node.

Write Pipeline Workflow

To write a file in HDFS, a client needs to interact with master i.e. namenode (master). Namenode provides the address of the datanodes (slaves) on which client will start writing the data. Client directly writes data on the datanodes, so datanode will create data write pipeline. The first datanode will copy the block to another datanode, which intern copy it to the third datanode. Once it creates the replicas of blocks, it sends back the acknowledgment.

As shown in the Figure 30 the data write operation in HDFS is distributed, client copies the data on datanodes, the steps by step explanation of data write operation is explained in next paragraph.

Step 1: The client creates the file by calling *create()* method on *DistributedFileSystem*.

Step 2: *DistributedFileSystem* makes an RPC call to the namenode to create a new file in the filesystem’s namespace, with no blocks associated with it. The namenode performs various checks to make sure the file doesn’t already exist and that the client has the right permissions to create the file. If these checks pass, the namenode makes a record of the new file; otherwise, file creation fails and the client is thrown an *IOException*. *TheDistributedFileSystem* returns an *FSDataOutputStream* for the client to start writing data to.

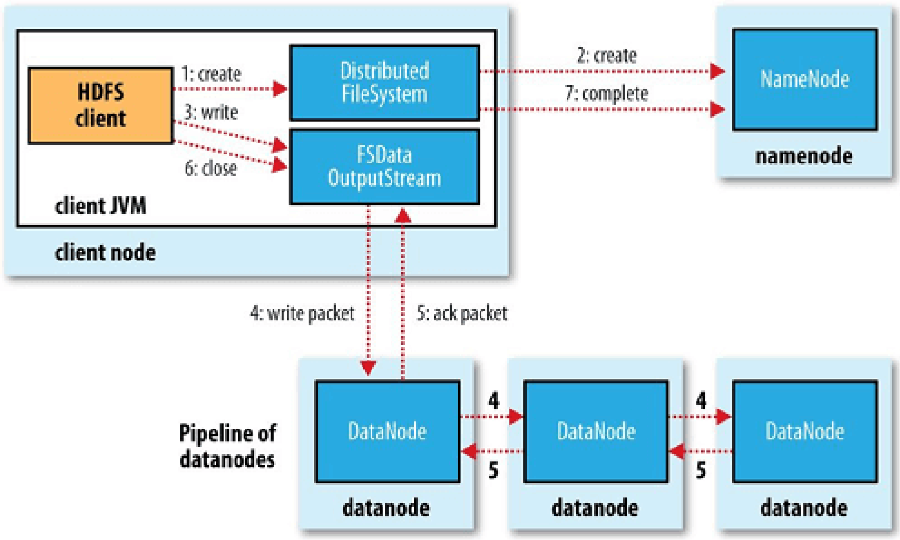


Figure 30 – Write Pipeline Workflow [99]

Step 3: As the client writes data, *DFSOutputStream* splits it into packets, which it writes to an internal queue, called the data queue. The data queue is consumed by the *DataStreamer*, which is responsible for asking the namenode to allocate new blocks by picking a list of suitable datanodes to store the replicas. The list of datanodes forms a pipeline, and here we'll assume the replication level is three, so there are three nodes in the pipeline. *TheDataStreamer* streams the packets to the first datanode in the pipeline, which stores the packet and forwards it to the second datanode in the pipeline.

Step 4: Similarly, the second datanode stores the packet and forwards it to the third (and last) datanode in the pipeline.

Step 5: DFSOutputStream also maintains an internal queue of packets that are waiting to be acknowledged by datanodes, called the *ack* queue. A packet is removed from the *ack* queue only when it has been acknowledged by all the datanodes in the pipeline.

Step 6: When the client has finished writing data, it calls close() on the stream.

Step 7: This action flushes all the remaining packets to the datanode pipeline and waits for acknowledgments before contacting the namenode to signal that the file is complete The namenode already knows which blocks the file is made up of, so it only has to wait for blocks to be minimally replicated before returning successfully [99].

Data write operation is considered successful if one replica is successfully written. It is governed by the property *dfs.namenode.replication.min* in *hdfs-default.xml* file. If there is any failure of datanode while writing a replica, the data written is not considered unsuccessful, but under-replicated which while balancing the cluster creates those missing replicas. Ack packet is independent of the status of data written to datanodes. Even if the data packet is not written the acknowledgement packet is delivered [100].

Generally, it's possible write data into HDFS from within JAVA code directly, sample code below [101] demonstrate this approach.

```java
FileSystem fileSystem = FileSystem.get(conf);

// Check if the file already exists
Path path = new Path("/path/to/file.ext");
if (fileSystem.exists(path)) {
  System.out.println("File " + dest + " already exists");
  return;
}

// Create a new file and write data to it.
FSDataOutputStream out = fileSystem.create(path);
InputStream in = new BufferedInputStream(new FileInputStream(new
File(source)));
byte[] b = new byte[1024];
int numBytes = 0;
while ((numBytes = in.read(b)) > 0) {
  out.write(b, 0, numBytes);
}

// Close all the file descripters
in.close();
out.close();
fileSystem.close();
```

3.1.5.4 HDFS Command Line

There is also another approach how to "push" data to HDFS and it's using command line. Before the command line examples will be explained, it's necessary to mention, that there're 3 more or less same command with tiny differences.

```
#  hadoop fs <args>   // this uses FsShell
#  hadoop dfs <args>  // this uses the now deprecated HDFS-specific DFSShell
#  hdfs dfs <args>    // replacement for hadoop dfs and recommended for HDFS
```

The `hadoop fs` relates to a generic file system which can point to any file systems like Local FS, S3 FS, HDFS etc. But `hadoop dfs` is very specific only to HDFS and was deprecated, now it's recommended to use `hdfs dfs` command instead.

It also beneficial to introduce most used commands used for filesystem operation. Nobody can't be surprised, that are very similar to linux equivalent.

1. **Create a directory** in HDFS at given path

```
   hadoop fs -mkdir <paths>
#  hadoop fs -mkdir /user/hadoop/wordcount
```

2. **List** the contents of a directory

```
   hadoop fs -ls <args>
#  hadoop fs -ls /user/hadoop
```

3. **Upload** and **download** a **file** in/from HDFS

Copy file from local file system to the Hadoop filesystem,

```
   hadoop fs -put <local_src_path> <hdfs_dest_path>
#  hadoop fs -put /home/hadoop/file.txt /user/hadoop/wordcount/
```

or vice versa.

```
   hadoop fs -get <hdfs_src_path> <local_dest_path>
#  hadoop fs -get /user/hadoop/wordcount/file.txt /home/hadoop
```

4. **See content** of a file

```
   hadoop fs -cat <filename>
#  hadoop fs -cat /user/hadoop/wordcount/file.txt
```

5. **Copy** a file from source to destination inside HDFS

```
   hadoop fs -cp <source> <dest>
#  hadoop fs -cp /user/hadoop/wordcount/file.txt /user/hadoop/dir2
```

6. Copy a file from/to local file system to HDFS

```
   hadoop fs -copyFromLocal <local_src_path> <hdfs_dest_path>
#  hadoop fs -copyFromLocal /home/hadoop/file.txt /user/hadoop
```

```
   hadoop fs -copyToLocal <hdfs_src_path> <local_dest_path>
#  hadoop fs -copytoLocal /user/hadoop/wordcount/file.txt /home/hadoop
```

7. **Move** file from source to destination

```
#  hadoop fs -mv <src> <dest>
#  hadoop fs -mv /user/hadoop/wordcount/file.txt /user/hadoop/dir2
```

8. **Remove** a **file or directory** in HDFS

```
   hadoop fs -rm <arg>
#  hadoop fs -rm /user/hadoop/wordcount/file.txt
```

9. **Display last few lines** of a file

```
   hadoop fs -tail <path[filename]>
#  hadoop fs -tail /user/hadoop/wordcount/file.txt
```

10. Display the aggregate **length** of a **file**

```
   hadoop fs -du <path>
#  hadoop fs -du /user/hadoop/wordcount/file.txt
>  size disk_space_consumed_with_all_replicas full_path_name
```

3.1.6 YARN

Apache YARN (Yet Another Resource Negotiator) is a component of Hadoop (from version 2.0) responsible for managing and allocating cluster resources. The over architecture of YARN and how is working is depicted in Figure 31.

Resource Manager

Resource Manager is the daemon responsible for global monitoring of resources (CPU, memory, network, disk) of the entire cluster and their allocation to applications. It includes an optional component called a scheduler that, if configured, provides resource allocation based on the rules defined by the administrator. Only one Resource Manager is active at a time and usually runs on Namenode.

Node Manager

Node Manager is a process running on each node in the cluster. He is responsible for monitoring his local resources. If this process is not running, it is not possible to run applications and calculations on the node (while Resource manager does not see the node's resources where Nod Manager is off). Its task is to allocate containers to applications (however, the application master does not have information about the application requirements, which is provided by the Application Master). Node Manager monitors the load on the containers (subsets of the application) - their CPU, memory, disks, etc., and reports the resource back to the Resource Manager. As a result, Resource Manager has the current state of resources available to other applications.

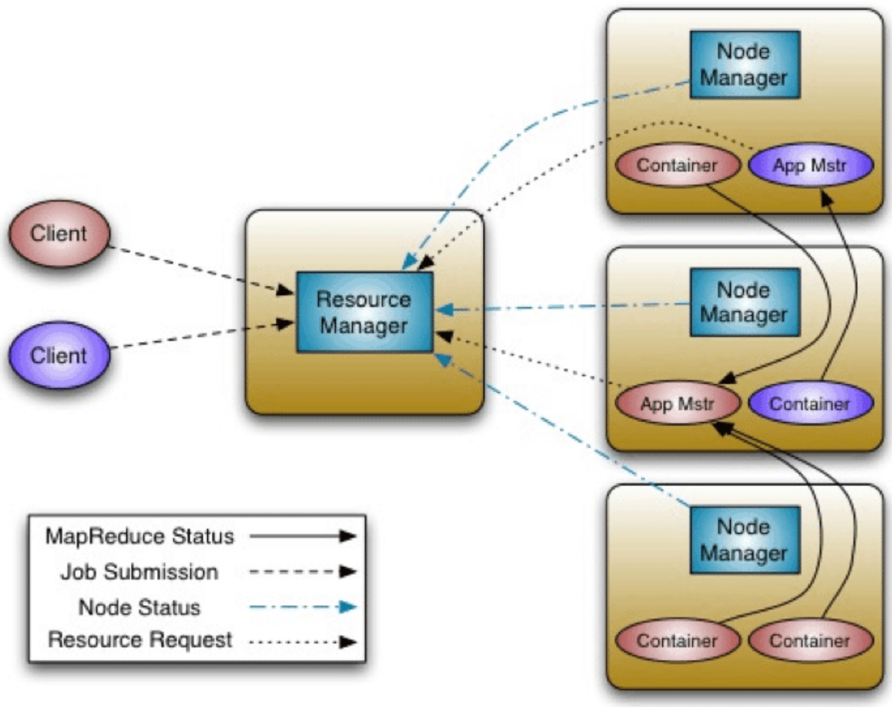


Figure 31 – YARN - How it works [102]

Application Master

Application Master is a process that is start when a client (user) sends an application for processing. Each application has its own Application Master - the first process (running in the first container) that is executed when the application is submitted for processing.

Application Master monitors the running of the application and gives feedback to the Resource Manager.

Capacity Scheduler

Fair Scheduler and Capacity Scheduler have been added to Hadoop since version 2.0. Capacity Scheduler ensures that a defined user group has a guaranteed cluster capacity. Fair Scheduler, if active, ensures that a group of users uses their capacity (containers) equally. For added flexibility, the Preemption feature was later added to release containers used by the running application if resources available to a group of users are less than the minimum guaranteed defined in Capacity Scheduler - the running application must release containers to use another application.

Container

Container is a YARN computing unit. Container is the place where individual application calculations (mappers) take place. Container has allocated a certain amount of CPU cores and RAM memory. The minimum and maximum container sizes can be limited in the *YARN.xml*. Several containers can run on one Datanode, the container cannot be moved to the next node. The Daemon responsible for allocating containers to the application is Node Manager.

3.1.7 MapReduce

MapReduce is a framework that allows large volumes of data to be processed in parallel on commodity hardware. This is a library written in Java. The idea behind the model is to divide a complex task or problem into smaller tasks. Reducing the complexity of processing smaller jobs is also less time consuming. In case of parallel data processing is used to complete such a task, the user initiating the task achieves much better speed of processing of the task. Parallelism has the great advantage of saving the time needed to complete a given task, but there are also issues to consider when designing a parallel data processing task. For example, the need to divide a complex task into several relatively equally large simpler tasks. However, such division may not always be unambiguous. And then further, the combination of results from smaller tasks may require additional tasks due to complexity, specifically dedicated to solving this problem and ultimately limiting the processing capacity of an individual server. The MapReduce programming model is designed so that the above problems are taken into account when processing tasks.

MapReduce algorithm combines two phases: *Map* and *Reduce*. In the *Map* part, the input data are divided into several parts (given by the number of mappers or files), where the calculation is performed over each of them. It can run on any cluster node. The Map phase transforms the data into a key-value and applies the logic written by the programmer (e.g. the sum of the same words). Then the *Reduce* phase is started, the input parameters are the pairs returned by the Map phase from each Datanodes. Reduce phase aggregates mapper results - merges all values with the same key and returns the result to the client.

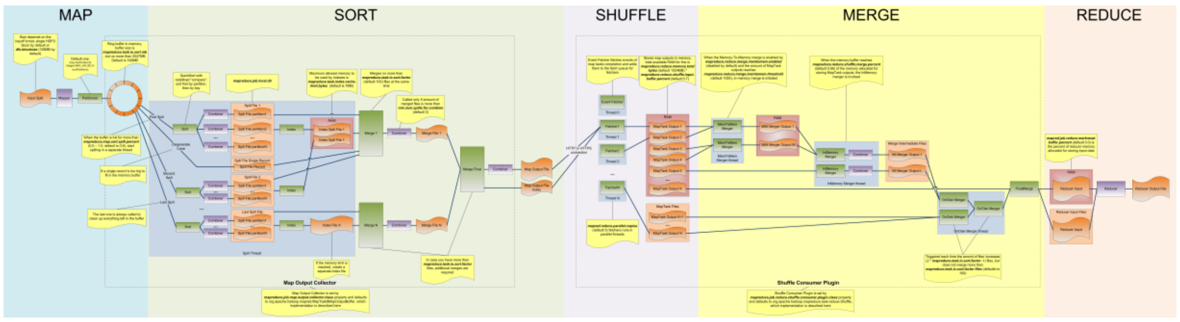


Figure 32 – Detail of MapReduce process [103]

However the MapReduce model is for simplicity divided into two basic phases, in reality consist of more phases. To support this statement please check or go directly online for hi-res picture.

Pros and Cons

The biggest advantage of MapReduce tasks is their scalability due to the **parallel processing**, where the job is divided among multiple nodes and each node works with a part of the job simultaneously. Once the task is written to run over few Datanodes, it can just equally run on thousands of Datanodes. Exactly this advantage has attracted a large number of developers, that there are no necessity to change code in case of significant increase the amount of nodes in the cluster. Second biggest advantage is **Data Locality** – when the data don't have to be transfer to processing unit, but the processing happened at data site.

On the other hand, the main disadvantage of MapReduce framework is that intermediate results from Map phase must still be stored in HDFS, which has a significant negative impact on the processing speed of MapReduce tasks. This is the biggest difference in the Spark framework, as will be mentioned below.

3.1.7.1 Examples of MapReduce usage

To illustrate how whole process of MapReduce works, example with file of animals will be used. Suppose, it's given Input file with words: "Deer Bear Dog Cat Dog Cat Deer Cat Bear".

First, the *input* file is divided into three *splits* as shown in the Figure 33. This will distribute the work among all the map nodes, where splits tokenize the words in each of the mapper and give a hardcoded value "1" to each of the words (tokens). A list of key-value pairs is created where the key is individual word and the value is hardcoded (1). So, for the first line (Deer Bear Dog) we have 3 key-value pairs – Deer, 1; Bear, 1; Dog, 1. The *mapping* process remains the same on all the nodes.

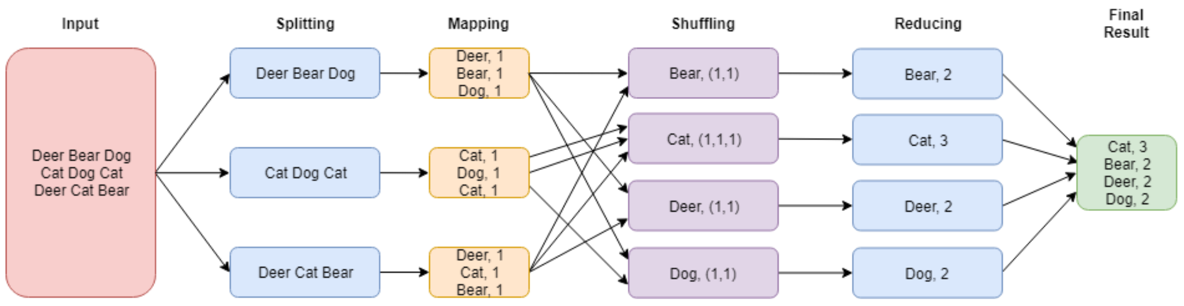


Figure 33 – Word Count Example of MapReduce

After *mapping* phase, *sorting and shuffling* happens so that all the tuples with the same key are sent to the corresponding *reducer*. So, after the sorting and shuffling phase, each reducer will have a unique key and a list of values corresponding to that very key. For example: <Bear, (1,1); Cat, (1,1,1); ...>. Now, each *reducer* counts the values which are present in that list of values. As shown in the Figure 33, *reducer* gets a list of values which is (1,1,1) for the key Cat. Then, it counts the number of 'ones' in the very list and gives the final output as <Cat, 3>.

Finally, all the output key/value pairs are then collected and written in the output file [104].

Let’s look closer on java implementation. In the first example, the most basic form will be shown. Afterward, the more advanced and professional piece of code will demonstrate the whole process of MapReduce application, together with comments placed right in the code to make the all code smooth to read and easy to understand. Generally, both examples (alike all implementations of MapReduce job) consist of main class *WordCount* with `main` method where all configuration of job is done. Next two important classes are *WordMapper* and *WordReducer* where the core of job is implemented. Now, please see below piece of code for full understanding of the process.

Example I – MapReduce WordCount – Basic Implementation

```
import java.io.IOException;
import java.util.StringTokenizer;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class WordCount {
  public static void main(String[] args) throws Exception {

    // Get the default configuration object and create the MapReduce job
    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "MapReduce WordCount");
    job.setJarByClass(WordCount.class);

    // tell Hadoop the mapper and the reducer to use
    job.setMapperClass(WordMapper.class);
    job.setCombinerClass(WordReducer.class);
    job.setReducerClass(WordReducer.class);

    // set output types for Key-Value
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);

    // set IO path
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));

    // submit the job and wait for it to complete!
    System.exit(job.waitForCompletion(true) ? 0 : 1);
  }

  public static class WordMapper extends Mapper<Object, Text, Text,
IntWritable> {
    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();
    public void map(Object key, Text value, Context context) throws
      IOException, InterruptedException {
      // tokenize line
      StringTokenizer itr = new StringTokenizer(value.toString());
      // make KV for each word
      while (itr.hasMoreTokens()) {
        word.set(itr.nextToken());
        context.write(word, one);
      }
    }
  }

  public static class WordReducer extends Reducer<Text, IntWritable, Text,
    IntWritable> {

    private IntWritable result = new IntWritable();

    public void reduce(Text key, Iterable<IntWritable> values, Context
context) throws IOException, InterruptedException {
      // total (sum) all values
      int total = 0;
      for (IntWritable val : values) {
        total += val.get();
      }
      result.set(total);
      // send it on to the reducer
      context.write(key, result);
    }
  }
}
```

Now, you have to create .jar file and run in on hadoop cluster

```
#   bin/hadoop jar $HADOOP_HOME/app/wordcount-I.jar WordCount
/user/hadoop/wordcount/hamlet.txt /user/hadoop/wordcount/hamlet-I.out
```

Details of job execution and output will be shown in next example.

Example II – MapReduce WordCount – Advanced Implementation

The second example is much more sophisticated, and all necessary comments are also included right in the code.

Class WordCount

```
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.IntWritable;
```

```java
public class WordCount {

  public static int main(String[] args) throws Exception {

    // if we got the wrong number of args, then exit
    if (args.length != 4 || !args[0].equals("-r")) {
      System.out.println("usage: WordCount -r <#reducers> <input> <output>");
      return -1;
    }

    // Get the default configuration object
    Configuration conf = new Configuration();

    // now create the MapReduce job
    Job job = Job.getInstance(conf, "MapReduce WordCount");

    // this tells Hadoop to ship around the jar file containing
    // "WordCount.class" to all of the different nodes so that they
    // can run the job
    job.setJarByClass(WordCount.class);

    // we'll output text/int pairs (since we have words as keys and
    // counts as values)
    job.setMapOutputKeyClass(Text.class);
    job.setMapOutputValueClass(IntWritable.class);

    // again we'll output text/int pairs (since we have words as keys and
    // counts as values)
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);

    // tell Hadoop the mapper and the reducer to use
    job.setMapperClass(WordCountMapper.class);
    job.setCombinerClass(WordCountReducer.class);
    job.setReducerClass(WordCountReducer.class);

    // we'll be reading in a text file, so we can use Hadoop's built-in
    // TextInputFormat
    job.setInputFormatClass(TextInputFormat.class);

    // we can use Hadoop's built-in TextOutputFormat for writing out
    // the output text file
    job.setOutputFormatClass(TextOutputFormat.class);

    // set the input and output paths
    TextInputFormat.setInputPaths(job, args[2]);
    TextOutputFormat.setOutputPath(job, new Path(args[3]));

    // set the number of reduce paths
    try {
      job.setNumReduceTasks(Integer.parseInt(args[1]));
    } catch (Exception e) {
      System.out.println("usage: WordCount -r <#reducers> <input> <output>");
      return -1;
    }

    // force the mappers to handle 50 kilobytes of input data each
    TextInputFormat.setMinInputSplitSize(job, 50 * 1024);
    TextInputFormat.setMaxInputSplitSize(job, 50 * 1024);
    // submit the job and wait for it to complete
    int exitCode = job.waitForCompletion(true) ? 0 : 1;
    return exitCode;

  }

}
```

Class WordCountMapper

```
import java.io.IOException;
import java.util.regex.Pattern;
import java.util.regex.Matcher;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

public class WordCountMapper extends Mapper<LongWritable, Text, Text,
IntWritable> {

  // create these for making below code clean
  private final static IntWritable one = new IntWritable(1);
  private Text word = new Text();

  // create a Pattern object to parse each line
  private final Pattern wordPattern = Pattern.compile("[a-zA-Z][a-zA-Z0-
9]+");

  public void map(LongWritable key, Text value, Context context) throws
    IOException, InterruptedException {

    // get a String version of the line
    String line = value.toString();

    // and a Matcher to parse it
    Matcher myMatcher = wordPattern.matcher(line);

    // while there are more tokens in the line
    while (myMatcher.find()) {

      // get the next pattern, and convert it to lower case
      String returnVal = myMatcher.group();
      returnVal = new String(returnVal.toLowerCase());
      word.set(returnVal);

      // send it on to the reducer
      context.write(word, one);
    }
  }
}
```

Class WordCountReducer

```
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

public class WordCountReducer extends Reducer<Text, IntWritable, Text,
IntWritable> {

  public void reduce(Text key, Iterable<IntWritable> values, Context context)
      throws IOException, InterruptedException {

    // loop through all of the counts that we got, adding them up
    int total = 0;
    for (IntWritable val : values) {
      total += val.get();
    }

    // and then send the counts to the reducer
    context.write(key, new IntWritable(total));
  }
}
```

Here, you can specify `-r` parameter with number of final reducers. In below outputs we'll go with just 1, otherwise the result would be flushed into "r" output files.

```
usage: WordCount -r <num reducers> <input> <output>

#   bin/hadoop jar $HADOOP_HOME/app/wordcount-II.jar WordCount -r 1
/user/hadoop/wordcount/hamlet.txt /user/hadoop/wordcount/hamlet-II.out
```

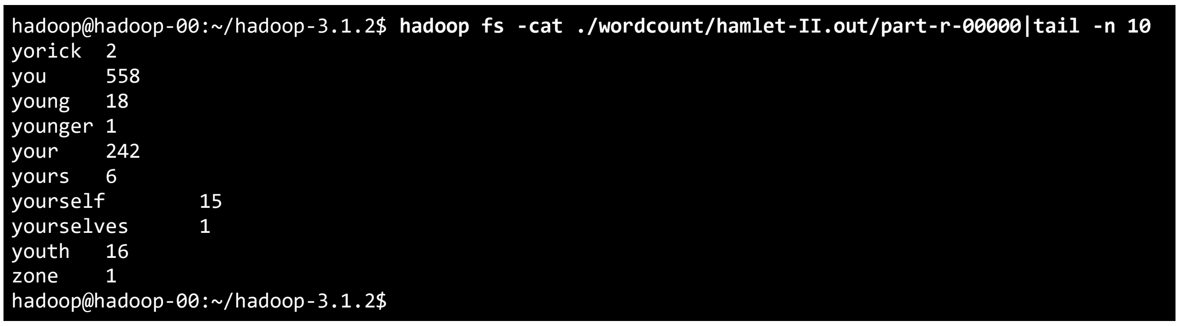
and you should expect output similar to this one

```
hadoop@hadoop-00:~/hadoop-3.1.2$ bin/hadoop jar $HADOOP_HOME/app/wordcount-II.jar WordCount -r 1
/user/hadoop/wordcount/hamlet.txt /user/hadoop/wordcount/hamlet-II.out
2019-05-02 11:06:07,518 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
2019-05-02 11:06:08,173 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not
performed. Implement the Tool interface and execute your application with ToolRunner to remedy
this.
2019-05-02 11:06:08,207 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding for path:
/tmp/hadoop-yarn/staging/hadoop/.staging/job_1556628279120_0012
2019-05-02 11:06:08,619 INFO input.FileInputFormat: Total input files to process : 1
2019-05-02 11:06:08,780 INFO mapreduce.JobSubmitter: number of splits:1
2019-05-02 11:06:09,139 INFO mapreduce.JobSubmitter: Submitting tokens for job:
job_1556628279120_0012
2019-05-02 11:06:09,141 INFO mapreduce.JobSubmitter: Executing with tokens: []
2019-05-02 11:06:09,508 INFO conf.Configuration: resource-types.xml not found
2019-05-02 11:06:09,511 INFO resource.ResourceUtils: Unable to find 'resource-types.xml'.
2019-05-02 11:06:09,627 INFO impl.YarnClientImpl: Submitted application
application_1556628279120_0012
2019-05-02 11:06:09,696 INFO mapreduce.Job: The url to track the job: http://hadoop-
00:8088/proxy/application_1556628279120_0012/
2019-05-02 11:06:09,697 INFO mapreduce.Job: Running job: job_1556628279120_0012
2019-05-02 11:06:21,896 INFO mapreduce.Job: Job job_1556628279120_0012 running in uber mode :
false
2019-05-02 11:06:21,903 INFO mapreduce.Job:  map 0% reduce 0%
2019-05-02 11:06:29,060 INFO mapreduce.Job:  map 100% reduce 0%
2019-05-02 11:06:37,195 INFO mapreduce.Job:  map 100% reduce 100%
2019-05-02 11:06:37,231 INFO mapreduce.Job: Job job_1556628279120_0012 completed successfully
2019-05-02 11:06:37,387 INFO mapreduce.Job: Counters: 53
        File System Counters
                FILE: Number of bytes read=60913
                FILE: Number of bytes written=555349
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=191853
                HDFS: Number of bytes written=42992
                HDFS: Number of read operations=8
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=2
        Job Counters
                Launched map tasks=1
                Launched reduce tasks=1
                Data-local map tasks=1
                Total time spent by all maps in occupied slots (ms)=5395
                Total time spent by all reduces in occupied slots (ms)=5177
                Total time spent by all map tasks (ms)=5395
                Total time spent by all reduce tasks (ms)=5177
                Total vcore-milliseconds taken by all map tasks=5395
                Total vcore-milliseconds taken by all reduce tasks=5177
                Total megabyte-milliseconds taken by all map tasks=5524480
                Total megabyte-milliseconds taken by all reduce tasks=5301248
        Map-Reduce Framework
                Map input records=4463
                Map output records=30847
                Map output bytes=282339
                Map output materialized bytes=60913
                Input split bytes=119
                Combine input records=30847
                Combine output records=4593
                Reduce input groups=4593
                Reduce shuffle bytes=60913
                Reduce input records=4593
                Reduce output records=4593
                Spilled Records=9186
                Shuffled Maps =1
                Failed Shuffles=0
                Merged Map outputs=1
                GC time elapsed (ms)=164
                CPU time spent (ms)=1940
                Physical memory (bytes) snapshot=355356672
                Virtual memory (bytes) snapshot=5187506176
                Total committed heap usage (bytes)=230821888
                Peak Map Physical memory (bytes)=229580800
                Peak Map Virtual memory (bytes)=2590453760
                Peak Reduce Physical memory (bytes)=125775872
                Peak Reduce Virtual memory (bytes)=2597052416
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=191734
        File Output Format Counters
                Bytes Written=42992
hadoop@hadoop-00:~/hadoop-3.1.2$
```

In the target directory, you can find min two files, where first one notes the result of job, and next file(-s) depend(s) on number of reducers. In case of more reduce jobs, the final result will be split into the same number of output files.

```
hadoop@hadoop-00:~/hadoop-3.1.2$ hadoop fs -ls ./wordcount/hamlet-II.out
Found 2 items
-rw-r--r--   3 hadoop supergroup          0 2019-05-02 11:06 wordcount/hamlet-II.out/_SUCCESS
-rw-r--r--   3 hadoop supergroup      42992 2019-05-02 11:06 wordcount/hamlet-II.out/part-r-00000
hadoop@hadoop-00:~/hadoop-3.1.2$
```

Just use `cat`, `head`, `tail` or simply combination of commands to see result of our job.



```
hadoop@hadoop-00:~/hadoop-3.1.2$ hadoop fs -cat ./wordcount/hamlet-II.out/part-r-00000|tail -n 10
yorick  2
you     558
young   18
younger 1
your    242
yours   6
yourself        15
yourselves      1
youth   16
zone    1
hadoop@hadoop-00:~/hadoop-3.1.2$
```

3.1.8 Summary

In this chapter the key components of the Hadoop framework has been discussed. The Hadoop core consists of HDFS, a distributed file system where data are stored. Data are stored in blocks. Each block is replicated according to the replication factor to other nodes of cluster. The blocks are divided into more racks, so they are not all in one rack if possible. Block metadata are stored by Namenode. MapReduce framework is a distributed computing model. The calculation takes place on multiple nodes in parallel. It has two phases. Map phase, where the data are tokenized and sorted in the key-value tuples and Reduce phase, where the Map phase results are aggregated. YARN is the COO (Chief Operations Officer) of the cluster. It takes care of resource monitoring, allocation and prioritization and also logs the entire process. It consists of a Resource Manager that monitors cluster resource usage. Node Manager takes care of monitoring server's local resources and allocating containers to applications. Application Master monitors the running of the application and gives feedback on the Resource Manager about used containers. Resource allocation can be planned by Capacity Scheduler, where you can configure queues and users who are entitled to them.

In the following sections will be introduced components or projects that are available in the Hadoop Ecosystem and make it easier for users to work with Hadoop. Will be described in more detail project like Spark and Sqoop, and other components that allow abstracting from Java application programming for the MapReduce model like Hive, where we use knowledge of SQL or Pig Latin in case of Pig.

3.2 General Data Processing

3.2.1 Apache Spark

The main drawback of the MapReduce model is that intermediate program results are written to disk, which is a major problem especially for quick response or iterative applications, as this approach makes them considerably slower and becomes a bottleneck for the entire system. Response to the shortcomings of the Hadoop MapReduce is Apache Spark. Spark is a unified analytics engine for large-scale data processing, but unlike Hadoop's MapReduce, it can minimize disk access by storing intermediate results in memory, so it is also suitable for interactive, exploratory analysis and iterative programs that accelerate against MapReduce up to one or two orders of magnitude.

In current version 2.4.2 (released on Apr 23 2019) Spark provides interfaces (API) for Java, Scala, Python, R and SQL. On May 8, 2019 the new version 2.4.3 has been released, unfortunately after deadline of this work. Here is a short summary of releases only for last 4 months:

Spark 2.4.3 released May 8, 2019

Spark 2.4.2 released April 23, 2019

Spark 2.4.1 released March 31, 2019

Spark 2.3.3 released February 15, 2019

Spark 2.2.3 released January 11, 2019

All these only demonstrate how fast R&D in this area especially with Apache Spark project is.

3.2.1.1 Spark Introduction

Apache Spark doesn't provide any storage (like HDFS) or any Resource Management capabilities. It is just a unified framework for processing large amount of data near to real time. In Figure 34, Apache Spark framework is organized in three major layers (listed from bottom to top.

1) *Resource Manager Layer* - As Apache Hadoop, Apache Spark doesn't comes up with Resource Management module like YARN. It manage resource in standalone mode in single node cluster setup. But for distributed cluster mode it can be integrated with resource management modules like YARN, Mesos or Kubernetess and can interact with many different data sources like HDFS, Alluxio, Cassandra, HBase, Hive and hundreds of other data sources [105].

2) *Spark Core Layer* - Spark Core is the base engine for large-scale parallel and distributed data processing. Further, additional libraries which are built on the top of the core allows diverse workloads for streaming, SQL, and machine learning. It is responsible for memory management and fault recovery, scheduling, distributing and monitoring jobs on a cluster & interacting with storage systems.

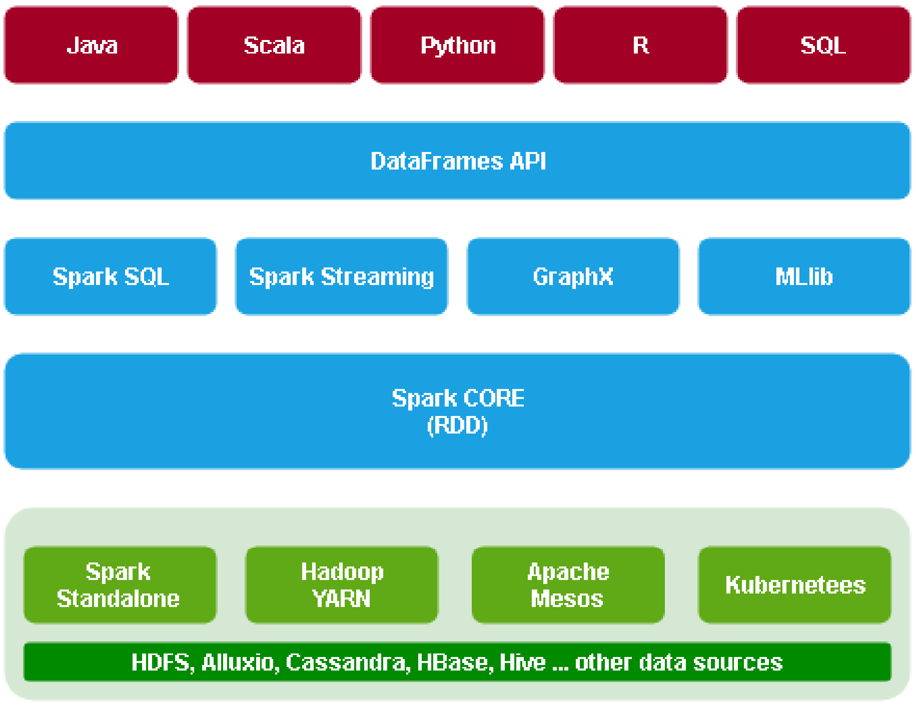


Figure 34 – Apache Spark Architecture

3) *Spark Ecosystem Layer* – consists of four key stack of libraries

- *Spark SQL* - module which integrates relational processing with Spark's functional programming API. It supports querying data either via SQL or via the HiveQL
- *Spark Streaming* - component which is used to process real-time streaming data
- *GraphX* - is a library for manipulating graphs and performing graph-parallel computations
- *MLlib* - used to perform common machine learning capabilities in Apache Spark MLlib provides multiple types of machine learning algorithms, including classification, regression, clustering, and collaborative filtering or supporting functionality such as model evaluation and data import. [106]

4) Last, but not considered as core layer, is an API for five languages like Java, Scala, Python, R and SQL.

Cluster Management Overview

Spark applications are run as independent sets of processes on a cluster, all coordinated by a central coordinator. This central coordinator can connect with three different cluster managers, Spark's Standalone, Apache Mesos, and Hadoop YARN (Yet Another Resource Negotiator).

When running an application in distributed mode on a cluster, Spark uses a master/slave architecture and the central coordinator, also called the driver program, is the main process in your application, running the code that creates a SparkContext object. This driver process is responsible for converting a user application into smaller execution units called tasks. These tasks are then executed by executors which are worker processes that run the individual tasks.

In a cluster, there is a master and any number of workers. The driver program, which can run in an independent process, or in a worker of the cluster, requests executors from the cluster manager. It then schedules the tasks composing the application on the executors obtained from the cluster manager. The cluster manager is responsible for the scheduling and allocation of resources across the host machines forming the cluster [107].

Resilient Distributed Dataset (RDD)

A Resilient Distributed Dataset (RDD) in Spark is simply an immutable distributed collection of records that is spread over one or more partitions that can be distributed and computed across different nodes of cluster. RDDs can contain any type of Python, Java, or Scala objects or user-defined classes. Users create RDDs in two ways: by loading an external data set, or by distributing a collection of objects in their driver program. RDDs are the building blocks of any Spark application and stand for:

- Resilient - fault tolerant and is capable of rebuilding data on failure
- Distributed - distributed data among the multiple nodes in a cluster
- Dataset - collection of partitioned data with values

It is a layer of abstracted data over the distributed collection. It is *immutable* (an object whose state cannot be modified after it is created) in nature and follows *lazy transformations* (the data inside RDD is not available or transformed until an action is). Indeed, transformations do not generate final values, they can be seen as instructions on how to manipulate with a data set, which Spark writes in a graph (Directed Acyclic Graph - DAG) while it passes the code. When an action is called on the RDD, Spark looks at that graph and then performs transformations that lead to materializing the data collection. By retaining the transformations, Spark can appropriately optimize data access that is actually accessed when it is needed. Last important feature – cacheable – means that you can hold all the data in a persistent "storage" like memory (default and the most preferred) or disk (the least preferred due to access speed)

RDDs offer two types of operations:

- **transformations** - construct a new RDD from a previous one; e.g. one common transformation is filtering data that matches a predicate; other operations could be: *map*, *flatMap*)
- **actions** - compute a result based on an RDD, and either return it to the driver program or save it to an external storage system (examples of operations: *collect*, *count*, *first*, *saveAsTextFile*)

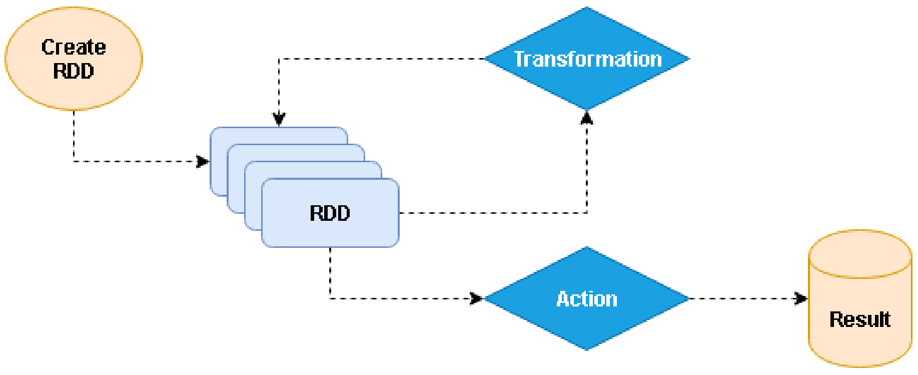


Figure 35 – RDD Operations

In below code snippet are presented some examples of transformations and action. For graphical representation please see Figure 35.

```
val txtFile = "hdfs://hadoop-01:9000/user/hadoop/wordcount/hamlet.txt"

val lineRDD = sc.textFile(txtFile)
//Transformation 1 -> DAG created
//{DAG: 1) Start -> [sc.textFile(txtFile)]}

val wordRDD = lineRDD.flatMap(_.split(" "))
//Transformation 2 -> wordRDD DAG updated
//{DAG: 1) Start -> [sc.textFile(logFile)]
//      2)       -> [lineRDD.flatMap(_.split(" "))]}

val filteredWordRDD = wordRDD.filter(_.equalsIgnoreCase("and"))
//Transformation 3 -> filteredWordRDD DAG updated
```

```
//{DAG: 1) Start -> [sc.textFile(logFile)]
//      2)       -> [lineRDD.flatMap(_.split(" "))]
//      3)       -> [wordRDD.filter(_.equalsIgnoreCase("and"))]}

filteredWordRDD.collect
//Action: collect
//Execute DAG & collect result to driver node
```

SparkSQL Highlighted

Spark SQL is a Spark module for structured (or semi-structured) data processing. Unlike the basic Spark RDD API, the interfaces provided by Spark SQL provide Spark with more information about the structure of both the data and the computation being performed. Internally, Spark SQL uses this extra information to perform extra optimizations. There are several ways to interact with Spark SQL including SQL and the Dataset API. When computing a result the same execution engine is used, independent of which API/language you are using to express the computation. This unification means that developers can easily switch back and forth between different APIs based on which provides the most natural way to express a given transformation. [108]

Datasets and DataFrames

A **Dataset** is a distributed collection of data. Dataset is a new interface added in Spark 1.6 that provides the benefits of RDDs (strong typing, ability to use powerful lambda functions) with the benefits of Spark SQL's optimized execution engine. A Dataset can be constructed from JVM objects and then manipulated using functional transformations (`map, flatMap, filter`, etc.). The Dataset API is available in Scala and Java [108].

A **DataFrame** is a Dataset organized into named columns. It is conceptually equivalent to a table in a relational database or a data frame in R/Python, but with richer optimizations under the hood. DataFrames can be constructed from a wide array of sources such as: structured data files, tables in Hive, external databases, or existing RDDs. The DataFrame API is available in Scala, Java, Python, and R. In Scala and Java, a DataFrame is represented by a Dataset of Rows. In the Scala API, DataFrame is simply a type alias of Dataset[Row]. While, in Java API, users need to use Dataset<Row> to represent a DataFrame [108].

3.2.1.2 Deploying Spark

Installation of Spark system consists of several steps. Here is a quick summary, and further each step is explained with commands/configuration and typical outputs of console. For installation will be used already installed server from Hadoop installation.

Apache Spark Installation Steps Overview

- Step 1: Ensure if Java is installed
- Step 2: Ensure if Scala is installed
- Step 3: Ensure if Git is installed
- Step 4: Download Spark
- Step 5: Install Spark
- Step 6: Verifying Spark installation

Step 1: Ensure if Java is installed

Before installing Spark, Java is a must have for your system. Following command will verify the version of Java.

```
# java -version
```

If Java is already installed on your system, you will see similar output to the following one, if not, please install Java (follow instruction in section 3.1.4.1 - Java Installation).

```
roman@hadoop-00:~$ java -version
openjdk version "1.8.0_191"
OpenJDK Runtime Environment (build 1.8.0_191-8u191-b12-2ubuntu0.18.04.1-b12)
OpenJDK 64-Bit Server VM (build 25.191-b12, mixed mode)
roman@hadoop-00:~$
```

Step 2: Ensure if Scala is installed

Installing Scala language is mandatory as well as Java before installing Spark as it is important for implementation. Scala language is used to implement Spark. Installation of Scala can be done with this command.

```
# sudo apt-get install scala
```

```
roman@hadoop-00:~$ sudo apt-get install scala
[sudo] password for roman:
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
...
update-alternatives: using /usr/share/scala-2.11/bin/scala to provide
/usr/bin/scala (scala) in auto mode
roman@hadoop-00:~$
```

So verify the Scala installation by using following command and write demo Hello World piece of code.

```
# scala
```

```
# scala> println("Hello World!")
```

```
roman@hadoop-00:~$ scala
Welcome to Scala 2.11.12 (OpenJDK 64-Bit Server VM, Java 1.8.0_191).
Type in expressions for evaluation. Or try :help.

scala> Hello World

scala>
```

As you could notice, you can't see what you're exactly writing in the Scala shell REPL. This is known bud in Ubuntu 18.04. The workaround is to uninstall Scala with `apt-get` and install again with `dpkg`. Please follow instructions below.

```
# sudo apt-get remove scala-library scala
# sudo wget www.scala-lang.org/files/archive/scala-2.12.8.deb
# sudo dpkg -i scala-2.12.8.deb
```

```
roman@hadoop-00:~$ sudo apt-get remove scala-library scala
... some output
roman@hadoop-00:~$ sudo wget www.scala-lang.org/files/archive/scala-2.12.8.deb
... some output
roman@hadoop-00:~$ sudo dpkg -i scala-2.12.8.deb
Selecting previously unselected package scala.
(Reading database ... 89038 files and directories currently installed.)
Preparing to unpack scala-2.12.8.deb ...
Unpacking scala (2.12.8-400) ...
Setting up scala (2.12.8-400) ...
Creating system group: scala
Creating system user: scala in scala with scala daemon-user and shell
/bin/false
Processing triggers for man-db (2.8.3-2ubuntu0.1) ...
roman@hadoop-00:~$ scala
Welcome to Scala 2.12.8 (OpenJDK 64-Bit Server VM, Java 1.8.0_191).
```

```
Type in expressions for evaluation. Or try :help.

scala> println("Hello World!")
Hello World!

scala> :q
roman@hadoop-00:~$
```

Step 3: Ensure if Git is installed

For installation of Spark it's requested Git installation, it can be done with this command. In our system, the Git is already installed, so the output will be skipped.

```
# sudo apt-get install git
```

Step 4: Download Spark

Next sub-step is download installation package for Spark from spark.apache.org site.

```
# wget https://archive.apache.org/dist/spark/spark-2.4.2/spark-2.4.2-bin-
hadoop2.7.tgz
```

Step 5: Install Spark

Once the download is done, extract it.

```
# tar xvf spark-2.4.2-bin-hadoop2.7.tgz
```

Step 6: Verifying Spark installation

Change the directory and run the Spark.

```
# cd spark-2.4.2-bin-hadoop2.7/bin/
# ./spark-shell
```

```
hadoop@hadoop-00:~/spark-2.4.2-bin-hadoop2.7/bin$ ./spark-shell
19/05/02 17:25:40 WARN NativeCodeLoader: Unable to load native-hadoop library
for your platform... using builtin-java classes where applicable
Using Spark's default log4j profile: org/apache/spark/log4j-
defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use
setLogLevel(newLevel).
Spark context Web UI available at http://hadoop-00:4040
Spark context available as 'sc' (master = local[*], app id = local-
1556817957360).
Spark session available as 'spark'.
Welcome to
```

```
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 2.4.2
      /_/

Using Scala version 2.12.8 (OpenJDK 64-Bit Server VM, Java 1.8.0_191)
Type in expressions to have them evaluated.
Type :help for more information.

scala>
```

and final test the Spark by using this command.

```
# println("Spark shell is running")
```

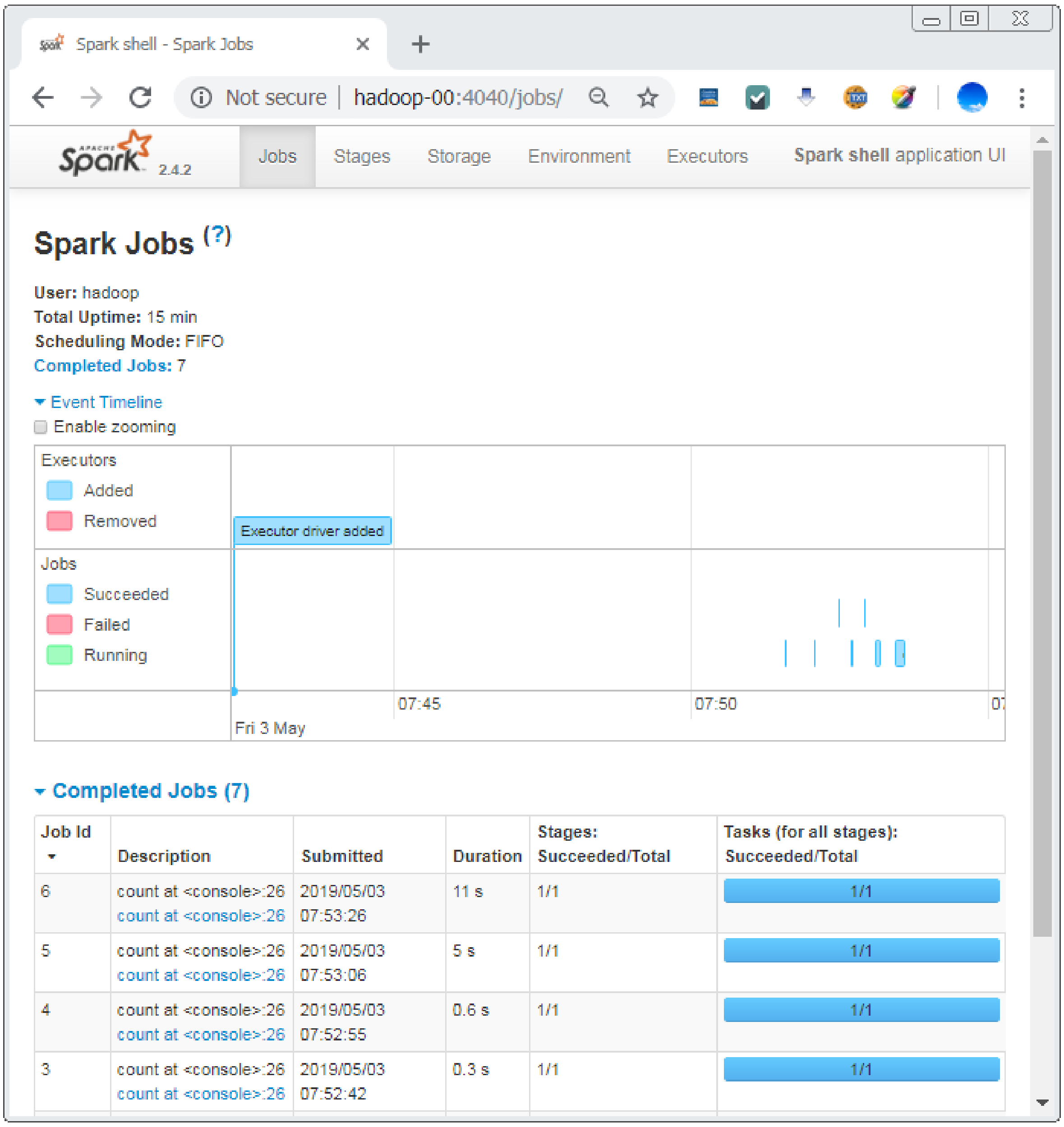
```
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 2.4.2
      /_/

Using Scala version 2.12.8 (OpenJDK 64-Bit Server VM, Java 1.8.0_191)
Type in expressions to have them evaluated.
Type :help for more information.

scala> println("Spark shell is running")
Spark shell is running

scala>
```

You can also check status of Spark resp. its jobs at Spark UI in web browser on port 4040 as shown in Figure 36.



Spark shell - Spark Jobs

Not secure | hadoop-00:4040/jobs/

Spark 2.4.2 | Jobs | Stages | Storage | Environment | Executors | **Spark shell** application UI

Spark Jobs (?)

User: hadoop
Total Uptime: 15 min
Scheduling Mode: FIFO
Completed Jobs: 7

▾ Event Timeline
☐ Enable zooming

▾ **Completed Jobs (7)**

Job Id ▾	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
6	count at <console>:26 count at <console>:26	2019/05/03 07:53:26	11 s	1/1	1/1
5	count at <console>:26 count at <console>:26	2019/05/03 07:53:06	5 s	1/1	1/1
4	count at <console>:26 count at <console>:26	2019/05/03 07:52:55	0.6 s	1/1	1/1
3	count at <console>:26 count at <console>:26	2019/05/03 07:52:42	0.3 s	1/1	1/1

Figure 36 – Spark UI on port 4040

3.2.1.3 Examples of Spark usage

Example 1 - Count of π

The easiest way how to run simple job on Spark is to calculate Pi. It doesn't need any interaction with filesystem or other components. In `scala-shell` run this short piece of code.

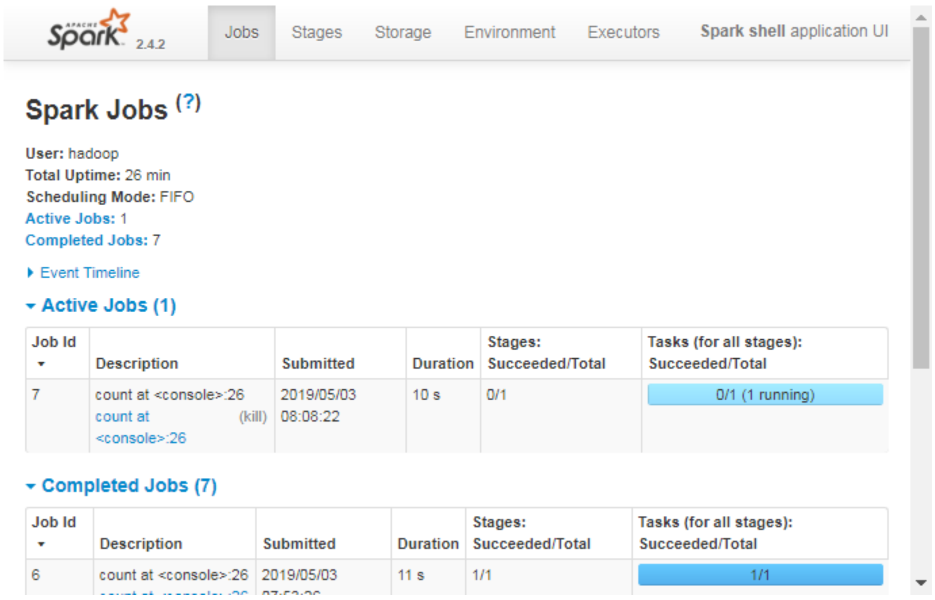
```
val NUM_SAMPLES = 100000000
val count = sc.parallelize(1 to NUM_SAMPLES).filter { _ =>
  val x = math.random
  val y = math.random
  x*x + y*y < 1
}.count()
println(s"Pi is roughly ${4.0 * count / NUM_SAMPLES}")
```

```
scala> val NUM_SAMPLES = 100000000
NUM_SAMPLES: Int = 100000000

scala> val count = sc.parallelize(1 to NUM_SAMPLES).filter { _ =>
     |   val x = math.random
     |   val y = math.random
     |   x*x + y*y < 1
     | }.count()
count: Long = 78539494

scala> println(s"Pi is roughly ${4.0 * count / NUM_SAMPLES}")
Pi is roughly 3.14157976
```

In between you can also check the active run of job in Spark Shell UI as shown in Figure 37.



Spark Jobs (?)

User: hadoop
Total Uptime: 26 min
Scheduling Mode: FIFO
Active Jobs: 1
Completed Jobs: 7

Event Timeline

Active Jobs (1)

Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
7	count at <console>:26 count at <console>:26 (kill)	2019/05/03 08:08:22	10 s	0/1	0/1 (1 running)

Completed Jobs (7)

Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
6	count at <console>:26	2019/05/03	11 s	1/1	1/1

Figure 37 – Spark Shell Active Job

Example 2 – Word count

In this example will be used the same data set as has been used in Hadoop MapReduce WordCount example. Also Spark will use Hadoop's HDFS to access the source data.

```
val textFile =
sc.textFile("hdfs://localhost:9000/user/hadoop/wordcount/hamlet.txt")
val words = textFile.flatMap(line => line.split(" "))
val wordsOne = words.map(word => (word, 1))
val wordsReduced = wordsOne.reduceByKey(_ + _)
val wordsSortedByValue = wordsReduced.sortBy(_._2)
wordsSortedByValue.saveAsTextFile("hdfs://localhost:9000/user/hadoop/wordcoun
t/hamlet.spark.out")
```

Output should be very similar to this one.

```
scala> val textFile =
sc.textFile("hdfs://localhost:9000/user/hadoop/wordcount/hamlet.txt")
textFile: org.apache.spark.rdd.RDD[String] =
hdfs://localhost:9000/user/hadoop/wordcount/hamlet.txt MapPartitionsRDD[38] at
textFile at <console>:24

scala> val words = textFile.flatMap(line => line.split(" "))
words: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[39] at flatMap at
<console>:25

scala> val wordsOne = words.map(word => (word, 1))
wordsOne: org.apache.spark.rdd.RDD[(String, Int)] = MapPartitionsRDD[40] at
map at <console>:25

scala> val wordsReduced = wordsOne.reduceByKey(_ + _)
wordsReduced: org.apache.spark.rdd.RDD[(String, Int)] = ShuffledRDD[41] at
reduceByKey at <console>:25

scala> val wordsSortedByValue = wordsReduced.sortBy(_._2)
wordsSortedByValue: org.apache.spark.rdd.RDD[(String, Int)] =
MapPartitionsRDD[44] at sortBy at <console>:25

scala>
wordsSortedByValue.saveAsTextFile("hdfs://localhost:9000/user/hadoop/wordcount
/hamlet.spark.out")

scala>
```

And finally we can check the result via `hadoop fs` commands. As below output shows, there's again similar structure of output directory, you can see one file {`_SUCCESS`} noted the status of job and several output result files (depending of number of reducers) {`part-00000`}.

```
hadoop@hadoop-00:~$ hadoop fs -ls ./wordcount/hamlet.spark.out
Found 2 items
-rw-r--r--   3 hadoop supergroup          0 2019-05-03 09:22
wordcount/hamlet.spark.out/_SUCCESS
-rw-r--r--   3 hadoop supergroup      89933 2019-05-03 09:22
wordcount/hamlet.spark.out/part-00000
hadoop@hadoop-00:~$
```

The result of job can be similarly seen using `hadoop fs -cat` command.

```
hadoop@hadoop-00:~$ hadoop fs -cat ./wordcount/hamlet.spark.out/part-00000
...
(a,453)
(I,523)
(to,608)
(of,625)
(and,680)
(the,929)
hadoop@hadoop-00:~$
```

As a last part of this example, the *action* on DAG is presented, without the need to output result to filesystem, so you can run another operation on top of it.

```
scala> val wordsSortedByValue = wordsReduced.sortBy(- _._2)
wordsSortedByValue: org.apache.spark.rdd.RDD[(String, Int)] = MapParti-
tionsRDD[48] at sortBy at <console>:25

scala> wordsSortedByValue.collect
res22: Array[(String, Int)] = Array((the,929), (and,680), (of,625), (to,608),
(I,523), (a,453), ...
```

Example 3 - Spark SQL

As you could notice, when the Spark's scala-shell is started, two object are created for you.

```
Spark context available as 'sc' (master = local[*], app id = local-155688729).
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 2.4.2
      /_/
```

The first one is *context* and the second one is *session*. `SparkSession` class ('spark' alias in console output above) is the entry point into all functionality in Spark. It can be created with code, but here, `scala-shell` have created it for you. All future lines of code wouldn't run without it. In next few snippets of code will be presented SQL access to data in Spark. With a `SparkSession`, applications can create DataFrames from an existing RDD, from a Hive table, or from Spark data sources. In out example, a DataFrame based on the content of a JSON file is created.

```
scala> val df = spark.read.json("examples/src/main/resources/people.json")
scala> df.show()
```

```
+----+-------+
| age|   name|
+----+-------+
|null|Michael|
|  30|   Andy|
|  19| Justin|
+----+-------+
```

As mentions earlier DataFrame provides a domain-specific language for structured and semi-structured data manipulation in Scala, Java, Python and R; i.e. data sets that you can specify a schema for. DataFrame is a tabular functional data abstraction or in other words just Dataset[Row] (data set of rows) in Scala and Java API.

In the following code snipped will be introduced some of structure query capabilities.

```
scala> df.printSchema()
root
 |-- age: long (nullable = true)
 |-- name: string (nullable = true)
scala> df.select("name").show()
+-------+
|   name|
+-------+
|Michael|
|   Andy|
| Justin|
+-------+
scala> df.filter($"age" > 21).show()
+---+----+
|age|name|
+---+----+
| 30|Andy|
+---+----+
scala> df.groupBy("age").count().show()
+----+-----+
| age|count|
+----+-----+
|  19|    1|
|null|    1|
|  30|    1|
+----+-----+
```

The `sql` function on a *SparkSession* enables applications to run SQL queries programmatically and returns the result as a DataFrame.

```
// Register the DataFrame as a SQL temporary view
scala> df.createOrReplaceTempView("people")

scala> val sqlDF = spark.sql("SELECT * FROM people WHERE age > 20")
scala> sqlDF.show()
```

```
+---+----+
|age|name|
+---+----+
| 30|Andy|
+---+----+
scala> val sqlDF = spark.sql("SELECT AVG(age) FROM people")
scala> sqlDF.show()
+--------+
|avg(age)|
+--------+
|    24.5|
+--------+
```

Temporary views in Spark SQL are session-scoped and will disappear if the session that creates it terminates. To have a temporary view that is shared among all sessions and keep alive until the Spark application terminates, you can create a *global temporary view* [105].

```
// Register the DataFrame as a global temporary view
scala> df.createGlobalTempView("people")

// Global temporary view is tied to a system preserved database `global_temp`
scala> spark.sql("SELECT * FROM global_temp.people").show()
+----+-------+
| age|   name|
+----+-------+
|null|Michael|
|  30|   Andy|
|  19| Justin|
+----+-------+
// Global temporary view is cross-session
spark.newSession().sql("SELECT * FROM global_temp.people").show()
+----+-------+
| age|   name|
+----+-------+
|null|Michael|
|  30|   Andy|
|  19| Justin|
+----+-------+
```

3.2.1.4 Summary

In this chapter the Apache Spark project, part of a Hadoop Ecosystem has been introduced. The Spark project is a unified analytics engine for large-scale data processing. Terms like RDD, DataFrames, Dataset and Transformation vs Action has been also discussed. Then the step-by-step of Spark deployment and configuration has been presented. In last section has been demonstrated how DAG is working and Spark's SQL access to data.

3.3 Data Analytics

Several tools are available to analyze data on the Hadoop platform. We introduce the two most used. They are Pig and Hive.

3.3.1 Apache Pig

3.3.1.1 Pig Introduction

Project called Pig started as a new research project Yahoo! in 2006. The goal of new project was to provide an alternative language interface to programming MapReduce using Java. Pig is considered a MapReduce abstraction as it detached the user from the underlying MapReduce code, enabling the developer to code in an interpreted data flow language, which is then converted to a series of map and reduce operations.

The scripting language used by Pig is called Pig Latin. The **workflow** of Pig Latin script processing looks like this or for graphical representation please see Figure 38:

1) an interpreter running on a client machine takes the Pig Latin instructions
2) turns these into a series of MapReduce jobs
3) submits these jobs to the cluster (or on single server)
4) monitors their progress
5) returns results to the console or saving results to files in HDFS

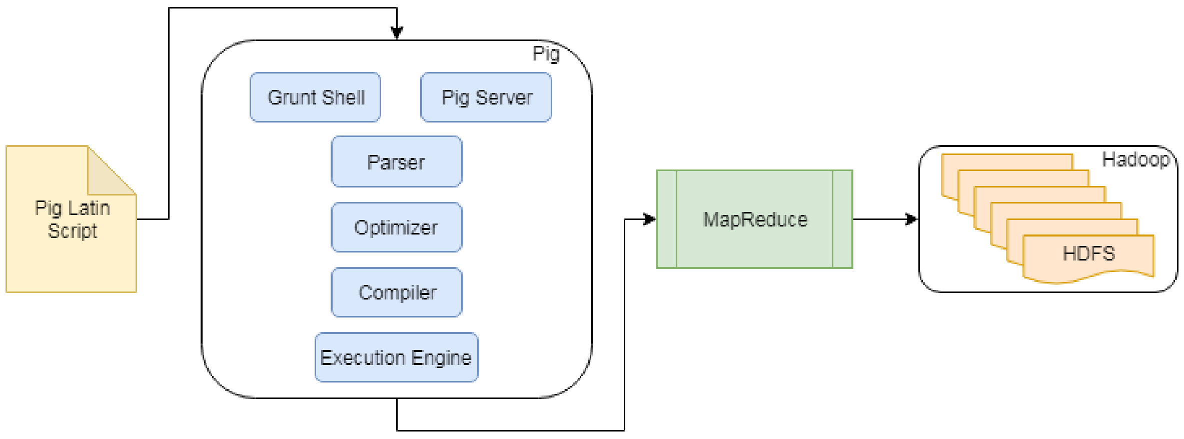


Figure 38 – Pig Latin Workflow

Among the **advantages** of Pig against MapReduce belong:

- developers don't need to know Java programming language

- Pig code does not need to be compiled and packaged, Java MapReduce must
- user with Pig don't have to describe objects, easier to data discovery

Pig Execution Modes

Pig has several modes it can be run in (execute Pig Latin statements and Pig commands). Pig can be run using `[pig]` command or `[java -cp pig.jar]`:

Local Mode `[pig -x local]` - Access to a single machine is needed; all files are installed and run using local host and file system

Tez Local Mode `[pig -x tez_local]` - It is similar to local mode, except internally Pig will invoke *tez* runtime engine. It's experimental mode as some errors issues interacting with bigger data.

Spark Local Mode `[pig -x spark_local]` - It is similar to local mode, except internally Pig will invoke *spark* runtime engine. It's experimental mode as some errors issues interacting with bigger data.

Mapreduce Mode `[pig OR pig -x mapreduce]` - Access to a Hadoop cluster and HDFS installation is expected for this mode. Mapreduce mode is the default mode.

Tez Mode `[pig -x tez]` - Access to a Hadoop cluster and HDFS installation is expected.

Spark Mode `[pig -x spark]` - To run Pig in Spark mode, you need access to a Spark, Yarn or Mesos cluster and HDFS installation.

The Pig Shell

Interactive mode – To run Pig in interactive mode using the Grunt shell just use the `[pig]` command (as shown below) and then you can continue with Pig Latin statements or other Pig commands.

```
hadoop@hadoop-00:~$ pig
grunt>
```

Batch Mode – You can run Pig also in batch mode using Pig scripts and the `[pig script.pig]` command.

Pig Latin

Pig Latin programs are a sequence of statements that follow the pattern described here.

- LOAD statement to read data from the file system into a named data set
- Series of "transformation" statements to process the data into a new data set
- DUMP statement to view results or STORE statement to save the results

Pig Data Structures

The data sets used by Pig are called **relations** or **bags** (from relational database point of view you can imagine as *tables*). Relations contain records called **tuples** (*rows*). Tuples contain **fields** (*columns*), and fields can contain other data structures (bags, tuples or atomic data called **atoms**). To understand differences please see Table 14 – Pig Data Structures.

Table 14 – Pig Data Structures

Type	Description	Example
relations / bags	collection of tuples	{(‘Marian’,31),(‘Erika’,28)}
tuple	set of fields	(‘Marian’,31)
field	a piece of data	‘Marian’
map	set of key value pairs	[name#Marian]

3.3.1.2 Deploying Pig

Step 1: Download latest Pig release

```
#  wget https://www-us.apache.org/dist/pig/pig-0.17.0/pig-0.17.0.tar.gz
```

```
hadoop@hadoop-00:~$ wget https://www-us.apache.org/dist/pig/pig-0.17.0/pig-
0.17.0.tar.gz
--2019-05-03 16:09:12--  https://www-us.apache.org/dist/pig/pig-0.17.0/pig-
0.17.0.tar.gz
Resolving www-us.apache.org (www-us.apache.org)... 40.79.78.1
Connecting to www-us.apache.org (www-us.apache.org)|40.79.78.1|:443...
connected.
HTTP request sent, awaiting response... 200 OK
Length: 230606579 (220M) [application/x-gzip]
Saving to: ‘pig-0.17.0.tar.gz’

pig-0.17.0.tar.gz   100%[==================>] 219.92M  2.34MB/s    in 95s

2019-05-03 16:10:48 (2.31 MB/s) - ‘pig-0.17.0.tar.gz’ saved
[230606579/230606579]

hadoop@hadoop-00:~$
```

Step 2: Unpack the Pig release

```
#  tar -xvf pig-0.17.0.tar.gz
```

Step 3: Verify Pig installation

```
#  export PIG_HOME=/home/hadoop/pig-0.17.0
#  export PATH=$PATH:$PIG_HOME/bin
```

```
hadoop@hadoop-00:~$ pig --version
Apache Pig version 0.17.0 (r1797386)
compiled Jun 02 2017, 15:41:58
hadoop@hadoop-00:~$
```

3.3.1.3 Examples of Pig usage

First, we have to prepare some data, which will be used in our examples.

```
#  wget http://hadoop.bistaff.eu/data/employees.csv
#  hadoop fs -mkdir emp
#  hadoop fs -put employees.csv emp
```

As mentioned earlier, typical Pig script consists of 3 stages (LOAD data, transformations, and DUMP/STORE data). All three stages now will be presented.

1) Loading Data into Pig

Loading data into a bag is the first step for any Pig program. To do this we can use three functions:

• PigStorage	`[PigStorage([delimit])]`	Loads and stores data as structured text files
• TextLoader	`[TextLoader()]`	Loads unstructured data in UTF-8 format
• JsonLoader	`[JsonLoader([schema])]`	Loads JSON data

```
#  grunt> emp = LOAD 'emp/employees.csv' USING PigStorage(',') AS (emp_id:int,
fname:chararray, lname:chararray, hire_date:chararray, salary:int);
```

It's good practice to immediately check:
- *the structure of bag*

```
#  grunt> DESCRIBE emp;
```

```
grunt> DESCRIBE emp;
emp: {emp_id: int,fname: chararray,lname: chararray,hire_date:
chararray,salary: int}
grunt>
```

- *sample of data* (command ILLUSTRATE physically touch the HDFS data)

```
#  grunt> ILLUSTRATE emp;
```

```
grunt> ILLUSTRATE emp;
...
-------------------------------------------------------------------------------------------------------
| emp    | emp_id:int    | fname:chararray    | lname:chararray    | hire_date:chararray    | salary:int    |
-------------------------------------------------------------------------------------------------------
|        | 131           | James              | Marlow             | 16/02/05               | 2500          |
-------------------------------------------------------------------------------------------------------
grunt>
```

or you can also use some of linux commands:

```
#  grunt> cat emp;
```

```
grunt> cat emp;
...
204,Hermann,Baer,07/06/02,10000
205,Shelley,Higgins,07/06/02,12008
206,William,Gietz,07/06/02,8300
grunt>
```

2) Transformations - Filtering, Projecting, and Sorting Data

In this phase, you can define several transformations. Remember, there's no action on particular command taken as all are done in last (third) phase, where the MapReduce job is submit, thus there's no valuable output to console.

First transformation is basic *projection* and datatype *transforming*. Projecting or transforming fields within tuples in bags in Pig is accomplished using the `FOREACH` statement.

```
#  grunt> emp_date = FOREACH emp GENERATE emp_id, UPPER(lname),
ToDate(hire_date, 'dd/MM/yy') AS hire_date, salary;
```

As you can see two built-in functions has been used (`UPPER, ToDate`). There many other built-in function available in Pig:

- Evaluation Functions	`AVG, COUNT, MAX, MIN, SIZE, SUM, TOKENIZE`
- Math Functions	`ABS, CEIL, EXP, FLOOR, LOG, RANDOM, ROUND`
- String Functions	`(STARTS|ENDS)WITH, LOWER/UPPER, (L|R|)TRIM, REGEX_EXTRACT`
- Datetime Functions	`CurrentTime, DaysBetween, Get(Day|Hour|Minute), ToDate`

We will again check result by issuing `ILLUSTRATE` command.

```
#  grunt> ILLUSTRATE emp_date;
```

```
grunt> ILLUSTRATE emp_date;
-------------------------------------------------------------------------------------------------------
| emp    | emp_id:int    | fname:chararray    | lname:chararray    | hire_date:chararray    | salary:int    |
-------------------------------------------------------------------------------------------------------
```

```
|        | 129           | Laura              | Bissot             | 20/08/05                 | 3300         |
------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------
| emp_date| emp_id:int | org.apache.pig.builtin.upper_lname_11:chararray | hire_date:datetime       | salary:int
|
--------------------------------------------------------------------------------------------------------------
|        | 129        | BISSOT                                          | 2005-08-20T00:00:00.000Z | 3300
--------------------------------------------------------------------------------------------------------------
grunt>
```

We can continue with another transformation like *filtering* with command `FILTER`:

```
#  grunt> emp_filtered = FILTER emp_date BY salary < 5000;
```

And last command in our example of transformations is final *ordering* with command `ORDER`:

```
#  grunt> emp_ordered = ORDER emp_filtered BY hire_date DESC;
```

3) Job submitting

And finally, we can run it. The whole workflow from Figure 38 will be performed and result (in this case) will be displayed in grunt console's output.

```
#  DUMP emp_ordered;
```

```
grunt> DUMP emp_ordered;
(128,MARKLE,2008-03-08T00:00:00.000Z,2200)
(136,PHILTANKER,2008-02-06T00:00:00.000Z,2200)
..
(137,LADWIG,2003-07-14T00:00:00.000Z,3600)
(115,KHOO,2003-05-18T00:00:00.000Z,3100)
grunt>
```

In last part of this example the output to HDFS instead of to console, will be presented.

a) Save the commands from previous part into file in Linux OS using e.g. `touch` and `nano` programs. Only last line (`DUMP` ...) replace with `STORE emp_ordered INTO 'emp_ordered';`.

```
#  touch $PIG_HOME/scripts/employees.pig
#  joe $PIG_HOME/scripts/employees.pig          // insert script here + save
```

```
emp = LOAD 'emp/employees.csv' USING PigStorage(',') AS (emp_id:int, fname:chararray, lname:chararray,
hire_date:chararray, salary:int);
emp_date = FOREACH emp GENERATE emp_id, UPPER(lname), ToDate(hire_date, 'dd/MM/yy') AS hire_date, salary;
emp_filtered = FILTER emp_date BY salary < 5000;
emp_ordered = ORDER emp_filtered BY hire_date DESC;
STORE emp_ordered INTO 'emp_ordered';
```

```
#  pig $PIG_HOME/scripts/employees.pig          // run the pig script
```

```
grunt hadoop@hadoop-00:~$ pig $PIG_HOME/scripts/employees.pig
HadoopVersion   PigVersion      UserId  StartedAt           FinishedAt          Features
3.1.2   0.17.0  hadoop  2019-05-04 11:15:21     2019-05-04 11:16:53     ORDER_BY,FILTER
```

```
Success!

Job Stats (time in seconds):
JobId   Maps    Reduces MaxMapTime      MinMapTime      AvgMapTime      MedianMapTime
MaxReduceTime   MinReduceTime   AvgReduceTime   MedianReducetime        Alias   Feature Outputs
job_1556868916740_0020  1       0       5       5       5       5       0       0       0       0
emp,emp_date,emp_filtered       MAP_ONLY
job_1556868916740_0021  1       1       5       5       5       5       5       5       5       5
emp_ordered     SAMPLER
job_1556868916740_0022  1       1       5       5       5       5       5       5       5       5
emp_ordered     ORDER_BY        hdfs://localhost:9000/user/hadoop/emp_ordered,

Input(s):
Successfully read 107 records (3912 bytes) from:
"hdfs://localhost:9000/user/hadoop/emp/employees.csv"

Output(s):
Successfully stored 49 records (2013 bytes) in: "hdfs://localhost:9000/user/hadoop/emp_ordered"

Counters:
Total records written : 49
Total bytes written : 2013
Spillable Memory Manager spill count : 0
Total bags proactively spilled: 0
Total records proactively spilled: 0

Job DAG:
job_1556868916740_0020  ->      job_1556868916740_0021,
job_1556868916740_0021  ->      job_1556868916740_0022,
job_1556868916740_
...
hadoop@hadoop-00:~$
```

The result can be again checked as we're used using `hadoop fs -cat` command.

```
#  hadoop fs -cat emp_ordered/part-r-00000 | tail -n 10
```

```
hadoop@hadoop-00:~$ hadoop fs -ls emp_ordered
Found 2 items
-rw-r--r--   3 hadoop supergroup    0 2019-05-04 11:16 emp_ordered/_SUCCESS
-rw-r--r--   3 hadoop supergroup 2013 2019-05-04 11:16 emp_ordered/part-r-00000

hadoop@hadoop-00:~$ hadoop fs -cat emp_ordered/part-r-00000 | tail -n 10
185     BULL     2005-02-20T00:00:00.000Z       4100
131     MARLOW   2005-02-16T00:00:00.000Z       2500
142     DAVIES   2005-01-29T00:00:00.000Z       3100
133     MALLIN   2004-06-14T00:00:00.000Z       3300
192     BELL     2004-02-04T00:00:00.000Z       4000
184     SARCHAND 2004-01-27T00:00:00.000Z       4200
141     RAJS     2003-10-17T00:00:00.000Z       3500
200     WHALEN   2003-09-17T00:00:00.000Z       4400
137     LADWIG   2003-07-14T00:00:00.000Z       3600
115     KHOO     2003-05-18T00:00:00.000Z       3100
hadoop@hadoop-00:~$
```

Word Count Example

```
grunt>
lines = LOAD '/user/hadoop/wordcount/hamlet.txt' AS (line:chararray);
words = FOREACH lines GENERATE FLATTEN(TOKENIZE(line)) AS word;
grouped = GROUP words BY word;
wordcount = FOREACH grouped GENERATE group AS word, COUNT(words) AS word_cnt;
```

```
ordered = ORDER wordcount BY word_cnt ASC, word DESC;
DUMP ordered;
```

```
...
(my,444)
(a,453)
(I,537)
(to,610)
(of,626)
(and,686)
(the,929)
grunt>
```

3.3.1.4 Summary

In this chapter the Apache Pig project, part of a Hadoop Ecosystem has been introduced. The Pig project is designed to abstract Java's MapReduce tasks to more familiar near SQL-like programming interface. How Pig works internally has been also mentioned. Then the step-by-step of Pig deployment and configuration has been presented. In last section has been demonstrated how to create and use Pig Latin scripts to access Hadoop's data.

3.3.2 Apache Hive

3.3.2.1 Hive Introduction

Hive is perhaps the most commonly used tool in the Hadoop ecosystem (followed closely by Spark and HBase). Similar to the Apache Pig project, introduced in previous chapter, which started at Yahoo!, the Apache Hive project started at Facebook in 2010 to provide a high-level interface to Hadoop MapReduce. Similar to Pig, the motivation for Hive was that Facebook realized that only few analysts were able to write MapReduce jobs in Java in contrast to general knowledge of SQL among them. Furthermore, SQL is the common language for business intelligence, visualization tools, and reporting tools, which commonly use ODBC/JDBC as a standard interface. Instead of creating a new language, as was done with PigLatin, the Hive project set out to put a SQL-like abstraction on top of MapReduce. The Hive project introduced a new language called HiveQL (or Hive Query Language), which implements a subset of SQL-92.

Workflow of processing HiveQL is very similar to Pig Latin workflow. Figure 39 provides a high-level depiction of how Hive processes data on HDFS. Workflow of HiveQL processing consist of following stages:

- executing query from the user interface
- parsed by the Hive client
- mapped to a sequence of Java MapReduce operations
- which are then submitted as jobs on the Hadoop cluster
- the progress would be monitored
- the results are returned
    - to the client or
    - written back to the desired location in HDFS.

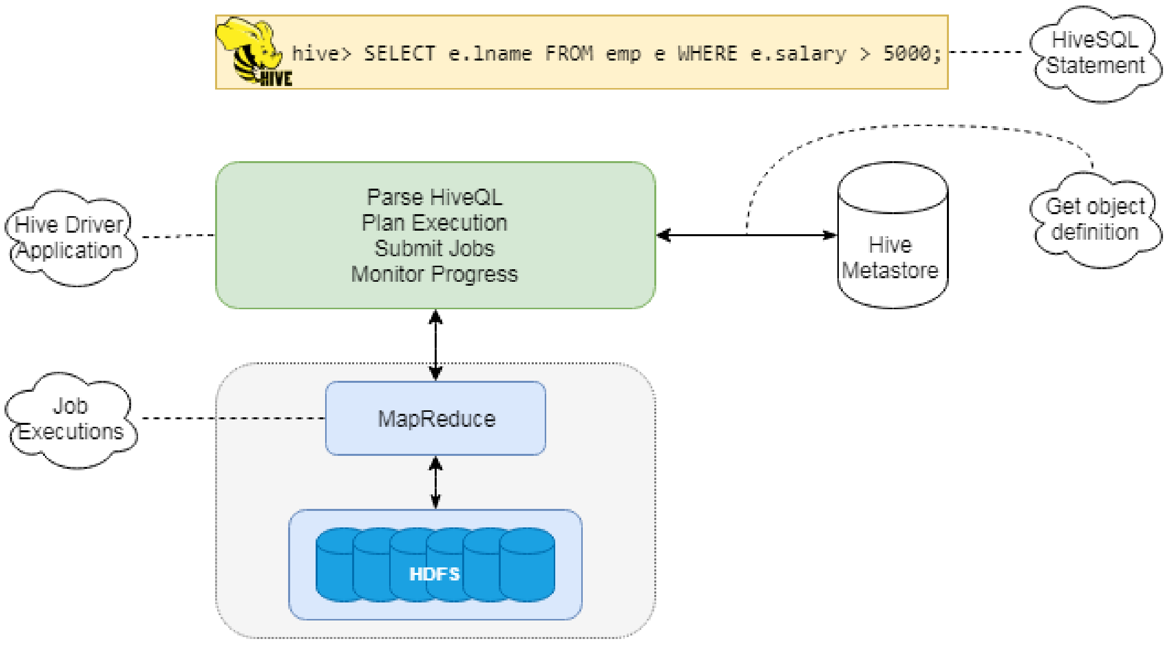


Figure 39 – HiveQL Workflow Process

3.3.2.2 Deploying Hive

In this section we will install and configure Hive within out pseudo-distributed Hadoop cluster using the default embedded Derby database which serve as Hive metadata storage.

Step 1: Prerequisites

Check if Java and Hadoop pseudo-distributed cluster is installed and well configured. If not, follow instructions from above chapters.

Step 2: Download Hive installation file

Depending on your release of Hadoop choose correct version of Hive. Apache Hive 3.1.1 release works with Hadoop 3.x.y.

```
#  wget https://www-us.apache.org/dist/hive/hive-3.1.1/apache-hive-3.1.1-
bin.tar.gz
```

```
hadoop@hadoop-00:~$ wget https://www-us.apache.org/dist/hive/hive-
3.1.1/apache-hive-3.1.1-bin.tar.gz
--2019-05-04 15:08:30--  https://www-us.apache.org/dist/hive/hive-
3.1.1/apache-hive-3.1.1-bin.tar.gz
Resolving www-us.apache.org (www-us.apache.org)... 40.79.78.1
Connecting to www-us.apache.org (www-us.apache.org)|40.79.78.1|:443...
connected.
HTTP request sent, awaiting response... 200 OK
Length: 280944629 (268M) [application/x-gzip]
Saving to: ‘apache-hive-3.1.1-bin.tar.gz’

apache-hive-3.1.1-bin.tar.gz
100%[==============================================================================
=========================================================>] 267.93M  2.38MB/s
in 1m 54s

2019-05-04 15:10:25 (2.36 MB/s) - ‘apache-hive-3.1.1-bin.tar.gz’ saved
[280944629/280944629]

hadoop@hadoop-00:~$
```

Step 3: Unzip the installation file

```
#  tar -xzvf apache-hive-3.1.1-bin.tar.gz
```

```
hadoop@hadoop-00:~$ tar -xzvf apache-hive-3.1.1-bin.tar.gz
```

Step 4: Create HIVE_HOME and set PATH

```
#  export HIVE_HOME=$HOME/apache-hive-3.1.1-bin
#  export PATH=$HIVE_HOME/bin:$PATH
```

Step 5: Make directories in HDFS for use by Hive, including the Hive warehouse directory

```
// directory tmp could already exists from hadoop installation, so just skip
#  hadoop fs -mkdir /tmp
#  hadoop fs -chmod g+w /tmp

#  hadoop fs -mkdir -p /user/hive/warehouse
#  hadoop fs -chmod g+w /user/hive/warehouse
```

Step 6: Create the metastore database

This will create metastore_db directory that contains Hive metadata. Hive Metastore is a database (by default MySQL, but we use Derby database as a part of distribution) where data about individual tables are stored (data storage location, table name, DDL definition, owner, scheme). Thanks to Metastore, the tables have a given structure and can be treated as relational tables. Objects stored in Metastore include:

- Database - namespace to which the table belongs

- Table - metadata about tables: table columns, name, owner, data location...

- Partition - information about partition name, storage location, columns.

```
#  cd $HIVE_HOME
#  schematool -initSchema -dbType derby
```

```
hadoop@hadoop-00:~$ cd $HIVE_HOME
hadoop@hadoop-00:~/apache-hive-3.1.1-bin$ schematool -initSchema -dbType derby
Metastore connection URL:
jdbc:derby:;databaseName=metastore_db;create=true
Metastore Connection Driver :    org.apache.derby.jdbc.EmbeddedDriver
Metastore connection User:       APP
Starting metastore schema initialization to 3.1.0
Initialization script hive-schema-3.1.0.derby.sql
...
Initialization script completed
schemaTool completed
hadoop@hadoop-00:~/apache-hive-3.1.1-bin$
```

If you get error like below, it means, you already have run hive, so it’s recommended to delete incomplete metastore with `mv` command and run `schematool` command once again.

```
hadoop@hadoop-00:~/apache-hive-3.1.1-bin$ schematool -initSchema -dbType derby
...
Starting metastore schema initialization to 3.1.0
Initialization script hive-schema-3.1.0.derby.sql


Error: FUNCTION 'NUCLEUS_ASCII' already exists. (state=X0Y68,code=30000)
org.apache.hadoop.hive.metastore.HiveMetaException: Schema initialization
FAILED! Metastore state would be inconsistent !!
Underlying cause: java.io.IOException : Schema script failed, errorcode 2
Use --verbose for detailed stacktrace.
*** schemaTool failed ***
hadoop@hadoop-00:~/apache-hive-3.1.1-bin$
```

```
#   mv metastore_db metastore_db.tmp
```

Step 7: Check the metastore database

```
#  hive> SHOW databases;
```

```
hadoop@hadoop-00:~$ hive
Hive Session ID = e8ba2856-e204-45a7-a5f0-9023c3827cc5
hive> SHOW databases;
OK
default
Time taken: 1.104 seconds, Fetched: 1 row(s)
hive>
```

Possible errors you could experienced

After creating *hive-site.xml* you could get **WstxParsingException: Illegal character entity: expansion character (code 0x8)** – solution is described in this link.

Another one could be **java.net.URISyntaxException** when starting HIVE and solution is described here.

3.3.2.3 *Examples of Hive usage*

In this example we will create two tables, ingest them with data and test some SQL commands.

Step 1: Open a Hive CLI

```
#  hive
```

```
hadoop@hadoop-00:~$ hive
Hive Session ID = e8ba2856-e204-45a7-a5f0-9023c3827cc5
hive>
```

Step 2: Create a new database called `bikeshare`

```
#  hive> CREATE DATABASE bikeshare;
```

```
hive> CREATE DATABASE bikeshare;
OK
Time taken: 0.303 seconds
hive> show databases;
OK
bikeshare
default
Time taken: 0.076 seconds, Fetched: 2 row(s)
hive>
```

Step 3: List the Hive databases available on your system

```
#  hive> SHOW databases;
```

```
hive> SHOW databases;
```

```
OK
bikeshare
default
Time taken: 0.076 seconds, Fetched: 2 row(s)
hive>
```

Step 4: Change your database context to the bikeshare database

```
#  hive> USE bikeshare;
```

```
hive> USE bikeshare;
OK
Time taken: 0.078 seconds
hive>
```

Step 5: Prepare data in HDFS

Download data to local filesystem.

```
#  wget https://s3.amazonaws.com/sty-hadoop/bike-share/stations/stations.csv
#  wget https://s3.amazonaws.com/sty-hadoop/bike-share/trips/trips.csv
```

Make directories in HDFS and upload both data sets.

```
#  hadoop fs -mkdir -p bikeshare/stations
#  hadoop fs -chmod 777 bikeshare/stations
#  hadoop fs -put stations.csv bikeshare/stations
#  hadoop fs -mkdir -p bikeshare/trips
#  hadoop fs -chmod 777 bikeshare/trips
#  hadoop fs -put trips.csv bikeshare/trips
```

Step 5: Create database tables

```
#  hive> CREATE EXTERNAL TABLE stations
           (
            station_id INT,
            name STRING,
            lat DOUBLE,
            long DOUBLE,
            dockcount INT,
            landmark STRING,
            installation STRING
           )
         ROW FORMAT DELIMITED
         FIELDS TERMINATED BY ','
         STORED AS TEXTFILE
         LOCATION 'bikeshare/stations';
```

```
hive> CREATE EXTERNAL TABLE stations
    >   (
    >    station_id INT,
    >    name STRING,
    >    lat DOUBLE,
    >    long DOUBLE,
    >    dockcount INT,
    >    landmark STRING,
    >    installation STRING
    >   )
    > ROW FORMAT DELIMITED
    > FIELDS TERMINATED BY ','
    > STORED AS TEXTFILE
    > LOCATION 'hdfs:///bikeshare/stations';
OK
Time taken: 1.011 seconds
hive>
```

Similarly create also second table.

```
#  hive> CREATE EXTERNAL TABLE trips
            (
             trip_id INT,
             duration INT,
             start_date STRING,
             start_station STRING,
             start_terminal INT,
             end_date STRING,
             end_station STRING,
             end_terminal INT,
             bike_num INT,
             subscription_type STRING,
             zip_code STRING
            )
         ROW FORMAT DELIMITED
         FIELDS TERMINATED BY ','
         STORED AS TEXTFILE
         LOCATION 'hdfs:///user/hadoop/bikeshare/trips';
```

```
hive> CREATE EXTERNAL TABLE trips
    >    (
    >     trip_id INT,
    >     duration INT,
    >     start_date STRING,
    >     start_station STRING,
    >     start_terminal INT,
    >     end_date STRING,
    >     end_station STRING,
    >     end_terminal INT,
    >     bike_num INT,
    >     subscription_type STRING,
    >     zip_code STRING
    >    )
    > ROW FORMAT DELIMITED
    > FIELDS TERMINATED BY ','
    > STORED AS TEXTFILE
    > LOCATION 'hdfs:///bikeshare/trips';
OK
Time taken: 0.2 seconds
hive>
```

As you might have noticed, two different location descriptions were used, at first table the relative path `LOCATION 'bikeshare/stations'` has been used and at second table full path with filesystem definition `LOCATION 'hdfs:///user/hadoop/bikeshare/trips'` has been used, both are valid and correct.

Step 6: List all tables in your database and look at their structure

```
#  hive> SHOW tables;
```

```
hive> SHOW tables;
OK
stations
trips
Time taken: 0.183 seconds, Fetched: 2 row(s)
hive>
```

```
#  hive> DESCRIBE stations;
```

```
hive> DESCRIBE stations;
OK
station_id              int
name                    string
lat                     double
long                    double
dockcount               int
landmark                string
installation            string
Time taken: 0.537 seconds, Fetched: 7 row(s)
hive>
```

```
#  hive> DESCRIBE FORMATTED stations;
```

```
hive> DESCRIBE FORMATTED stations;
OK
# col_name              data_type               comment
station_id              int
name                    string
lat                     double
long                    double
dockcount               int
landmark                string
installation            string

# Detailed Table Information
Database:               bikeshare
OwnerType:              USER
Owner:                  hadoop
CreateTime:             Sat May 04 16:32:17 UTC 2019
LastAccessTime:         UNKNOWN
Retention:              0
Location:               hdfs://localhost:9000/bikeshare/stations
Table Type:             EXTERNAL_TABLE
Table Parameters:
        EXTERNAL                TRUE
        bucketing_version       2
        numFiles                1
        totalSize               5214
        transient_lastDdlTime   1556987537

# Storage Information
SerDe Library:          org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe
InputFormat:            org.apache.hadoop.mapred.TextInputFormat
OutputFormat:
org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat
Compressed:             No
Num Buckets:            -1
Bucket Columns:         []
Sort Columns:           []
Storage Desc Params:
        field.delim             ,
        serialization.format    ,
Time taken: 0.667 seconds, Fetched: 36 row(s)
hive>
```

Step 7: Test first SQLs

Let’s start with simple SQL and count number of records in table.

```
#  hive> SELECT COUNT(1) FROM stations;
```

```
hive> SELECT COUNT(1) FROM stations;
Query ID = hadoop_20190505094534_e18ce1ae-9c48-4475-890f-46b581a95e65
Total jobs = 1
...
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 4.43 sec   HDFS Read: 18517
HDFS Write: 102 SUCCESS
Total MapReduce CPU Time Spent: 4 seconds 430 msec
OK
vvv RESULT of SQL vvv
70
^^^ RESULT of SQL ^^^
Time taken: 34.008 seconds, Fetched: 1 row(s)
hive>
```

In next example we continue with some projection, selection, and grouping.

```
#  hive> SELECT start_terminal
               ,start_station
               ,COUNT(1) AS count
           FROM trips
          GROUP BY start_terminal, start_station
          ORDER BY count
           DESC LIMIT 10;
```

```
hive> SELECT start_terminal
    >       ,start_station
    >       ,COUNT(1) AS count
    >   FROM trips
    >  GROUP BY start_terminal, start_station
    >  ORDER BY count
    >   DESC LIMIT 10;
Query ID = hadoop_20190505095144_9222fea6-54e8-45ac-9669-e4f42445f20b
Total jobs = 2
...
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 5.66 sec   HDFS Read:
43027008 HDFS Write: 3200 SUCCESS
Stage-Stage-2: Map: 1  Reduce: 1   Cumulative CPU: 4.03 sec   HDFS Read: 11192
HDFS Write: 567 SUCCESS
Total MapReduce CPU Time Spent: 9 seconds 690 msec
OK
vvv RESULT of SQL vvv
70      San Francisco Caltrain (Townsend at 4th)      26304
69      San Francisco Caltrain 2 (330 Townsend)       21758
50      Harry Bridges Plaza (Ferry Building)          17255
55      Temporary Transbay Terminal (Howard at Beale) 14436
60      Embarcadero at Sansome                        14158
61      2nd at Townsend                               14026
65      Townsend at 7th                               13752
74      Steuart at Market                             13687
67      Market at 10th                                11885
77      Market at Sansome                             11431
Time taken: 77.526 seconds, Fetched: 10 row(s)
hive>
```

And finally we try JOIN operation on both tables.

```
#  hive> SELECT t.trip_id
               ,t.duration
               ,t.start_date
               ,s.name
           FROM stations s
           JOIN trips t ON s.station_id = t.start_terminal;
          LIMIT 10;
```

```
hive> SELECT t.trip_id
    >       ,t.duration
    >       ,t.start_date
    >       ,s.name
    >   FROM stations s
    >   JOIN trips t ON s.station_id = t.start_terminal
    >  LIMIT 10;
Query ID = hadoop_20190505101406_5ec44fc8-bfc1-465e-8785-f25aad228d42
Total jobs = 1
Execution completed successfully
MapredLocal task succeeded
Launching Job 1 out of 1
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_1556868916740_0047, Tracking URL = http://hadoop-
00:8088/proxy/application_1556868916740_0047/
Kill Command = /home/hadoop/hadoop-3.1.2/bin/mapred job  -kill
job_1556868916740_0047
Hadoop job information for Stage-3: number of mappers: 1; number of reducers:
0
```

```
2019-05-05 10:14:38,903 Stage-3 map = 0%,  reduce = 0%
2019-05-05 10:14:49,526 Stage-3 map = 100%,  reduce = 0%, Cumulative CPU 3.2
sec
MapReduce Total cumulative CPU time: 3 seconds 200 msec
Ended Job = job_1556868916740_0047
MapReduce Jobs Launched:
Stage-Stage-3: Map: 1   Cumulative CPU: 3.2 sec   HDFS Read: 883065 HDFS
Write: 723 SUCCESS
Total MapReduce CPU Time Spent: 3 seconds 200 msec
OK
vvv RESULT of SQL vvv
913460  765     8/31/2015 23:26 Harry Bridges Plaza (Ferry Building)
913459  1036    8/31/2015 23:11 San Antonio Shopping Center
913455  307     8/31/2015 23:13 Post at Kearney
913454  409     8/31/2015 23:10 San Jose City Hall
913453  789     8/31/2015 23:09 Embarcadero at Folsom
913452  293     8/31/2015 23:07 Yerba Buena Center of the Arts (3rd @ Howard)
913451  896     8/31/2015 23:07 Embarcadero at Folsom
913450  255     8/31/2015 22:16 Embarcadero at Sansome
913449  126     8/31/2015 22:12 Beale at Market
913448  932     8/31/2015 21:57 Post at Kearney
^^^ RESULT of SQL ^^^
Time taken: 44.634 seconds, Fetched: 10 row(s)
hive>
```

Data Output with Hive

Most of the time, it's requested to persist the result of SQL to a local or distributed filesystem - specifically HDFS. Hive supports several methods to accomplish this, including the following:

- `INSERT OVERWRITE` – export the query results to another Hive table and overwrite the existing contents
- `INSERT INTO TABLE` – export the query results to another Hive table and appends the output to the existing contents
- `INSERT OVERWRITE DIRECTORY` – export the results to a directory in HDFS (file / Hive table)
- `INSERT OVERWRITE LOCAL DIRECTORY` – export the results to a local directory, not to HDFS

3.3.2.4 Summary

In this chapter the Apache Hive project, part of a Hadoop Ecosystem has been introduced. The Hive project is designed to enable fast and simple access to data in HDFS using a familiar SQL-like programming interface. How Hive works internally, including what is the purpose of metastore, that stores object definitions for Hive tables has been also mentioned. Then the step-by-step of Hive deployment and configuration has been presented. In last section has been demonstrated how to create and access Hive tables using both Hive DDL (Data Definition Language) and DML (Data Manipulation Language) statements.

3.4 Data Ingestion

Apache Hadoop with its storage system HDFS is just pure storage, as well as RDBMS are storage for their data. If we want to work with data in Hadoop, it's necessary to create them or import them from external system. In chapter HDFS Command Line has been introduced how it is possible to import data into HDFS from local filesystem using command line `hadoop fs -put`. In this chapter will be presented how to import/export data between relational databases and HDFS with Apache Sqoop.

3.4.1 Apache Sqoop

3.4.1.1 Sqoop Introduction

Apache Sqoop is a data ingestion tool designed for efficiently transferring bulk data between structured data-stores such as relational databases and Apache Hadoop (as shown in Figure 40), Apache Hive and Apache HBase. Sqoop support the transfer between RDBMS and Hadoop bi-directional, to Hive and HBase supports only unidirectional transfers.

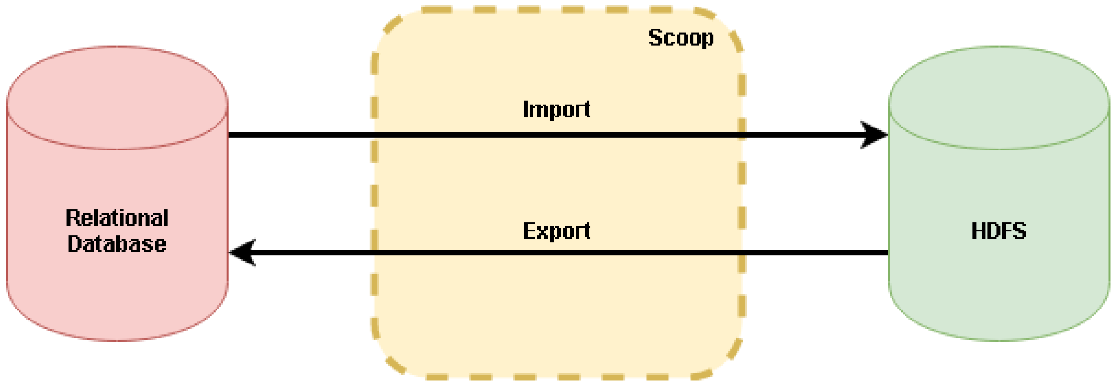


Figure 40 – Apache Sqoop Import/Export

Sqoop Import

Sqoop can be used for importing data from a relational database into HDFS. The input to the import process is a database table. Sqoop reads the table row-by-row and store it into HDFS. The output of this import process is a set of files containing a copy of the imported table. The import process is performed in parallel. For this reason, the output will be in multiple files.

Sqoop Export

Sqoop can be used for exporting data from a HDFS into relational database. Sqoop reads the set of delimited text files from HDFS in parallel, parses them into records, and inserts them as new rows in a target database table.

3.4.1.2 Deploying Sqoop

In this section we will install and configure Hive within out pseudo-distributed Hadoop cluster using the default embedded Derby database which serve as Hive metadata storage.

Step 1: Prerequisites

Check if **Java** and **Hadoop** pseudo-distributed cluster is installed and well configured. If not, follow instructions from above chapters.

Step 2: Download Sqoop installation file

```
#  wget https://www-us.apache.org/dist/sqoop/1.4.7/sqoop-1.4.7.bin__hadoop-
2.6.0.tar.gz
```

Step 3: Unzip Sqoop installation file

```
#  tar -xvf sqoop-1.4.7.bin__hadoop-2.6.0.tar.gz
```

Step 4: Create SQOOP_HOME and set PATH

```
#  export SQOOP_HOME=$HOME/sqoop-1.4.7.bin__hadoop-2.6.0
#  export PATH=$SQOOP_HOME/bin:$PATH
```

Step 5: Configure Sqoop

If `HADOOP_COMMON_HOME` and `HADOOP_MAPRED_HOME` wasn't set with Hadoop installation, make following changes:

```
#  cd $SQOOP_HOME/conf
#  mv sqoop-env-template.sh sqoop-env.sh
```

Open `sqoop-env.sh` and edit the following lines:

```
export HADOOP_COMMON_HOME=$HADOOP_HOME
export HADOOP_MAPRED_HOME=$HADOOP_HOME
```

Step 6: Download and Configure mysql-connector-java

```
#  wget http://ftp.ntu.edu.tw/MySQL/Downloads/Connector-J/mysql-connector-
java_8.0.16-1ubuntu18.04_all.deb
#  sudo apt install ./mysql-connector-java_8.0.16-1ubuntu18.04_all.deb
#  mv mysql-connector-java-8.0.16.jar $SQOOP_HOME/lib
```

Step 7: Verifying Sqoop

```
#  sqoop-version
```

```
hadoop@hadoop-00:~$ sqoop-version
...
2019-05-05 15:42:11,602 INFO sqoop.Sqoop: Running Sqoop version: 1.4.7
Sqoop 1.4.7
git commit id 2328971411f57f0cb683dfb79d19d4d19d185dd8
Compiled by maugli on Thu Dec 21 15:59:58 STD 2017
hadoop@hadoop-00:~$
```

3.4.1.3 Examples of Sqoop usage

In the following example will be described the process of importing table from MySQL relational database to HDFS. Is it expected, that MySQL database is already installed, configured and populated with some data.

Step 1: Check MySQL source data

To start interactive shell for MySQL database, run command `mysql`.

```
#  mysql
```

```
hadoop@hadoop-00:~$ mysql
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 33
```

```
Server version: 5.7.26-0ubuntu0.18.04.1 (Ubuntu)

Copyright (c) 2000, 2019, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

Query the list of local databases and use one.

```
#  SHOW databases;
```

```
mysql> SHOW databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sys                |
| test               |
+--------------------+
5 rows in set (0.00 sec)

mysql> USE test
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql>
```

And finally check the data.

```
#  select * from my_emp limit 5;
```

```
mysql> select * from my_emp limit 5;
+--------+-----------+---------+---------------------+--------+
| emp_id | fname     | lname   | hire_date           | salary |
+--------+-----------+---------+---------------------+--------+
|    100 | Steven    | King    | 2003-06-17 00:00:00 |  24000 |
|    101 | Neena     | Kochhar | 2005-09-21 00:00:00 |  17000 |
|    102 | Lex       | De Haan | 2001-01-13 00:00:00 |  17000 |
|    103 | Alexander | Hunold  | 2006-01-03 00:00:00 |   9000 |
|    104 | Bruce     | Ernst   | 2007-05-21 00:00:00 |   6000 |
+--------+-----------+---------+---------------------+--------+
5 rows in set (0.00 sec)

mysql>
```

Step 1: Check MySQL source data

To start interactive shell for MySQL database, run command `mysql`.

```
#  mysql
```

```
hadoop@hadoop-00:~$ mysql
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 33
Server version: 5.7.26-0ubuntu0.18.04.1 (Ubuntu)

Copyright (c) 2000, 2019, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

Step 2: Import data via Sqoop from MySQL to HDFS

To start interactive shell for MySQL database, run command `mysql`.

```
#  sqoop import --connect "jdbc:mysql://localhost/test" --username root --
table my_emp --target-dir mysql/my_emp
```

```
hadoop@hadoop-00:~$ sqoop import --connect "jdbc:mysql://localhost/test" --
username root --table my_emp --target-dir mysql/my_emp
...
2019-05-05 15:06:36,967 INFO db.DBInputFormat: Using read commited transaction
isolation
2019-05-05 15:06:36,968 INFO db.DataDrivenDBInputFormat: BoundingValsQuery:
SELECT MIN(`emp_id`), MAX(`emp_id`) FROM `my_emp`
2019-05-05 15:06:36,974 INFO db.IntegerSplitter: Split size: 26; Num splits: 4
from: 100 to: 206
...
2019-05-05 15:06:49,110 INFO mapreduce.Job: Job job_1556868916740_0052 running
in uber mode : false
2019-05-05 15:06:49,114 INFO mapreduce.Job:  map 0% reduce 0%
2019-05-05 15:07:12,409 INFO mapreduce.Job:  map 25% reduce 0%
2019-05-05 15:07:15,430 INFO mapreduce.Job:  map 50% reduce 0%
2019-05-05 15:07:16,438 INFO mapreduce.Job:  map 75% reduce 0%
2019-05-05 15:07:17,463 INFO mapreduce.Job:  map 100% reduce 0%
2019-05-05 15:07:17,503 INFO mapreduce.Job: Job job_1556868916740_0052
completed successfully
2019-05-05 15:07:17,638 INFO mapreduce.Job: Counters: 33
        File System Counters
                FILE: Number of bytes read=0
                FILE: Number of bytes written=900776
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=441
                HDFS: Number of bytes written=4818
                HDFS: Number of read operations=24
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=8
        Job Counters
                Killed map tasks=1
                Launched map tasks=4
                Other local map tasks=4
                Total time spent by all maps in occupied slots (ms)=88670
                Total time spent by all reduces in occupied slots (ms)=0
                Total time spent by all map tasks (ms)=88670
                Total vcore-milliseconds taken by all map tasks=88670
                Total megabyte-milliseconds taken by all map tasks=90798080
...
        File Output Format Counters
                Bytes Written=4818
2019-05-05 15:07:17,652 INFO mapreduce.ImportJobBase: Transferred 4.7051 KB in
43.5016 seconds (110.7545 bytes/sec)
2019-05-05 15:07:17,661 INFO mapreduce.ImportJobBase: Retrieved 107 records.
hadoop@hadoop-00:~$
```

From the output, you can read, that first Sqoop query for min and max values of primary key and split read data to 4 part, so 4 jobs were running and import data into 4 equally parts in HDFS.

Step 3: Check data in HDFS

At the end, we can check data right in the HDFS with well-known command.

```
#  hadoop fs -ls mysql/my_emp
#  hadoop fs -cat mysql/my_emp/part-m-00000
#  hadoop fs -cat mysql/my_emp/part-m-00001
#  hadoop fs -cat mysql/my_emp/part-m-00002
#  hadoop fs -cat mysql/my_emp/part-m-00003
```

```
hadoop@hadoop-00:~$ hadoop fs -ls mysql/my_emp
Found 5 items
-rw-r--r-- 3 hadoop supergroup    0 2019-05-05 15:07 mysql/my_emp/_SUCCESS
-rw-r--r-- 3 hadoop supergroup 1216 2019-05-05 15:07 mysql/my_emp/part-m-00000
-rw-r--r-- 3 hadoop supergroup 1211 2019-05-05 15:07 mysql/my_emp/part-m-00001
-rw-r--r-- 3 hadoop supergroup 1172 2019-05-05 15:07 mysql/my_emp/part-m-00002
-rw-r--r-- 3 hadoop supergroup 1219 2019-05-05 15:07 mysql/my_emp/part-m-00003
hadoop@hadoop-00:~$ hadoop fs -cat mysql/my_emp/part-m-00000
100,Steven,King,2003-06-17 00:00:00.0,24000
101,Neena,Kochhar,2005-09-21 00:00:00.0,17000
...
125,Julia,Nayer,2005-07-16 00:00:00.0,3200
126,Irene,Mikkilineni,2006-09-28 00:00:00.0,2700
hadoop@hadoop-00:~$ hadoop fs -cat mysql/my_emp/part-m-00001
127,James,Landry,2007-01-14 00:00:00.0,2400
128,Steven,Markle,2008-03-08 00:00:00.0,2200
...
152,Peter,Hall,2005-08-20 00:00:00.0,9000
153,Christopher,Olsen,2006-03-30 00:00:00.0,8000
hadoop@hadoop-00:~$ hadoop fs -cat mysql/my_emp/part-m-00002
154,Nanette,Cambrault,2006-12-09 00:00:00.0,7500
155,Oliver,Tuvault,2007-11-23 00:00:00.0,7000
```

```
...
178,Kimberely,Grant,2007-05-24 00:00:00.0,7000
179,Charles,Johnson,2008-01-04 00:00:00.0,6200
hadoop@hadoop-00:~$ hadoop fs -cat mysql/my_emp/part-m-00003
180,Winston,Taylor,2006-01-24 00:00:00.0,3200
181,Jean,Fleaur,2006-02-23 00:00:00.0,3100
...
205,Shelley,Higgins,2002-06-07 00:00:00.0,12008
206,William,Gietz,2002-06-07 00:00:00.0,8300
hadoop@hadoop-00:~$
```

There is also possibility to see HDFS data with one command.

```
#  hadoop fs -cat mysql/my_emp/part-m-*
```

```
hadoop@hadoop-00:~$ hadoop fs -cat mysql/my_emp/part-m-00000
100,Steven,King,2003-06-17 00:00:00.0,24000
101,Neena,Kochhar,2005-09-21 00:00:00.0,17000
...
...
...
...
...
...
...
...
205,Shelley,Higgins,2002-06-07 00:00:00.0,12008
206,William,Gietz,2002-06-07 00:00:00.0,8300
hadoop@hadoop-00:~$
```

3.4.1.4 Summary

In this chapter the Apache Sqoop project, part of a Hadoop Ecosystem has been introduced. The Sqoop project is designed to transfer data between relational databases and HDFS in both direction. Step-by-step of Sqoop deployment and configuration has been presented. In the last section has been demonstrated how to make an import of relation table from MySQL database to HDFS.

CONCLUSION

Human differs from all other species on this planet, there's no doubt about it. The start of using sounds and voices led to better communication. Sharing knowledge and experience with descendants and next generations brought our genus to the necessity of recording information in a persistent form. First writing systems appeared 6000 BC and this is generally considered as milestone when human accelerate the speed of evolution. Ability to easy communicate, record and share thoughts and information started new epoch of human – The age of information. Another big milestone of recording and sharing information came in 1450 AC with Gutenberg's printing press. It took next 500 years for another invention in human information processing – the first computers. And only next 50 years brought us to the Age of Internet. Each of these four milestones exponentially increased the amount of information processed.

This thesis deals with phenomenon of *Big Data* and is divided into 3 big chapters. In the first chapter branded "Big Data" I started with explanation of human information processing from the very beginning and went through the whole millennia until these days. In very next subchapter I explained the difference or the hierarchy of Data-Information-Knowledge-Wisdom, so every reader of this work has basic understanding why do we process the data. The business and technical requests in the age of internet brought IT experts to challenge of processing of unbelievable volume, velocity and variety of data. Old relational database systems were not able to address these requests.

All these bring us to the second chapter branded *NoSQL*. Exactly, NoSQL databases was the right answer to this challenge. In the beginning of this chapter I presented these technologies from various viewpoints. Explanation of distribution and consistency of data leads to presenting CAP theorem. I continued with general classification of database systems from different perspectives and finished with a high-level taxonomy of the NoSQL datastores based on the data model which can classify them into five major categories: key-value stores, document stores, wide-column (column-oriented) stores, graph databases and multi-model databases. In the rest of this chapter I took a closer look on each of these types of NoSQL databases.

In the third chapter (practical part) branded *Apache Hadoop Ecosystem* I started with presenting Apache Hadoop project itself - the system for distributed storage and processing of enormous amount of data. Distributed filesystem and Resource manager are described from big picture perspective into details. The processing model (MapReduce) continues with first set of examples and use cases. In the rest of practical part, I continued with presentation of projects (technologies) from Hadoop Ecosystem, like Apache Spark – for general data processing, Apache Pig and Apache Hive – for data access and analytics and finishing with Apache Sqoop – technology for data ingestion. Integral part of every single technology in this chapter is detail step-by-step guide how to install and configure particular technology. For each chosen technology I provided a set of functional examples and use cases with explanation how each of them work and fit together. Short summaries are attached at the end for each particular technology.

Although one might think that it is a young technology or area of technologies, definitely the biggest challenge for me was the form of picking up the most important and present it in compressed way for different level of knowledge of readers. The biggest value of this work I see in collecting, summarization and presentation of phenomenon of Big Data and NoSQL technologies in one comprehensive paper. The practical part I built as stack of technologies, so the step-by-step instructions guide reader with installation and configuration and follow with explanation of typical example scenarios. It should be noted that only really core and most used technologies were presented in this work as the whole ecosystem consists of dozens of projects.

At one place the reader gets all necessary and comprehensive introduction into the field of Big Data and NoSQL. Although the work is meant to be read from top down, different level of readers are free to jump right into the chapter or technology of their interests.

Continuation of this thesis could be in horizontal direction and studying the other projects from Hadoop Ecosystem or in vertical direction, where honored reader can dig deeper in particular technology. Author’s intention is to focus on Data Science and Machine Learning during his further self-education.

If I dare to subjectively evaluate the results of the work, I accomplished all submission points of the thesis and I truly believe, that also all goals I have defined in the beginning of this work have been met.

BIBLIOGRAPHY

[1] DAY, Lance and Ian MCNEIL. *Biographical Dictionary of the History of Technology.* Florence: Taylor and Francis, 2005. ISBN 978-0-203-02829-2.

[2] FITCH, Tecumseh W. The evolution of speech: a comparative review. Trends in Cognitive Sciences. *Trends in Cognitive Sciences.* 2000, Vol. 4, No. 7. DOI: 10.1016/S1364-6613(00)01494-7.

[3] POWELL, Barry B. *Writing : Theory and History of the Technology of Civilization.* New York: John Wiley & Sons, 2012. ISBN 978-1-118-25532-2.

[4] MAYER-SCHÖNBERGER, Viktor and Kenneth CUKIER. *Big Data: A Revolution That Will Transform How We Live, Work, and Think.* Boston : Mariner Books, 2014. ISBN 978-0-544-22775-0.

[5] From Data to Wisdom: The Path of the Most Successful Investors. *Value Walk.* [Online] Apr 26, 2017. [Cited: May 1, 2019]. Available from: https://www.valuewalk.com/2017/04/data-wisdom-path-successful-investors/

[6] VISHWAKARMA, Ashish. Difference between Structured, Semi-structured and Unstructured data. *geeksforgeeks.org.* [Online] [Cited: May 1, 2019]. Available from: https://www.geeksforgeeks.org/difference-between-structured-semi-structured-and-unstructured-data

[7] KALYVAS, James R. and Michael R. OVERLY. *Big Data: A Business and Legal Guide.* Boca Raton, Florida: CRC Press, 2015. ISBN 978-1-4665-9237-7.

[8] What is the DIKW Pyramid? *Ontotext.* [Online] [Cited: May 1, 2019]. Available from: https://www.ontotext.com/knowledgehub/fundamentals/dikw-pyramid/

[9] SCOTT, Jim. How to Evolve from RDBMS to NoSQL + SQL. *MapR.* [Online] Feb 11, 2016. [Cited: May 1, 2019]. Available from: https://mapr.com/blog/how-evolve-rdbms-nosql-sql/

[10] CHANG, Fay, Jeffrey DEAN, Sanjay GHEMAWAT. Bigtable: A Distributed Storage System for Structured Data. *Google AI.* [Online] 2006. [Cited: May 1, 2019]. Available from: https://ai.google/research/pubs/pub27898

[11] DECANDIA, Giuseppe, Deniz HASTORUN and Madan JAMPANI. Dynamo: Amazon's Highly Available Key-Value Store. *All Things Distributed.* [Online] 2007.

[Cited: May 1, 2019]. Available from: http://www.allthingsdistributed.com/files/amazon-dynamo-sosp2007.pdf

[12] NoSQL - A Relational Database Management System. [Online] [Cited: May 1, 2019]. Available from: http://www.strozzi.it/cgi-bin/CSA/tw7/I/en_US/nosql

[13] SADALAGE, Pramod and Martin FOWLER. *NoSQL Distilled : A Brief Guide to the Emerging World of Polyglot Persistence.* Upper Saddle River: Addison-Wesley, 2012. ISBN 978-0-321-82662-6.

[14] YUHANNA, Noel, Gene LEGANZA and Christian AUSTIN. The Forrester Wave™: Big Data NoSQL. *MapR.* [Online] Aug 17, 2016. [Cited: May 1, 2019]. Available from: http://go.mapr.com/rs/846-BMC-777/images/forrester-nosql-wave-2016-define-your-nosql-strategy.pdf

[15] How Much Data Is Generated Per Minute. *Digital Information World.* [Online] 2018. [Cited: May 1, 2019]. Available from: https://www.digitalinformationworld.com/2018/06/infographics-data-never-sleeps-6.html

[16] Big Data: el uso de los datos para saber todo. *Deusto.* [Online] Oct 18, 2018. [Cited: May 1, 2019]. Available from: https://blogs.deusto.es/master-informatica/wp-content/uploads/2018/10/need.jpeg

[17] History of Stored Data. *Gavroshe USA.* [Online] [Cited: May 1, 2019]. Available from: https://gavroshe.com/history-of-stored-data/

[18] REINSEL, David, John GANTZ and John RYDNING. Data Age 2025. *The Digitization of the World.* [Online] November 2018. [Cited: May 1, 2019]. Available from: https://www.seagate.com/files/www-content/our-story/trends/files/idc-seagate-dataage-whitepaper.pdf

[19] KLEIN, Andy. Hard Drive Cost Per Gigabyte. *backblaze.com.* [Online] Jul 17, 2017. [Cited: May 1, 2019]. Available from: https://www.backblaze.com/blog/hard-drive-cost-per-gigabyte/

[20] KOMOROWSKI, Matt. A history of storage cost. *MKomo.* [Online] Mar 9, 2014. [Cited: May 1, 2019]. Available from: http://www.mkomo.com/cost-per-gigabyte-update.

[21] LANEY, Doug. 3D Data Management: Controlling Data Volume, Velocity, and Variety. *www.gartner.com.* [Online] Feb 6, 2001. [Cited: May 1, 2019]. Available from:

https://blogs.gartner.com/doug-laney/files/2012/01/ad949-3D-Data-Management-Controlling-Data-Volume-Velocity-and-Variety.pdf

[22] Big Data: Definitions and Concepts. *Big Data : Concepts, Challenges and Solutions.* [Online] [Cited: May 1, 2019]. Available from: http://bigdata-tech.blogspot.com/p/big-data-definitions-and-concepts.html

[23] The Four V's of Big Data. *IBM Big Data & Analytics Hub.* [Online] [Cited: May 1, 2019]. Available from: https://www.ibmbigdatahub.com/sites/default/files/styles/xlarge-scaled/public/infographic_image/4-Vs-of-big-data.jpg

[24] What is Big Data? *BigData.black.* [Online] May 21, 2016. [Cited: May 1, 2019]. Available from: http://bigdata.black/featured/what-is-big-data/

[25] Big Data Technology with 8 V´s. *M Brain.* [Online] [Cited: May 1, 2019]. Available from: http://m-brain.com

[26] Big Data Visualization. *Datamation.* [Online] Jul 7, 2017. [Cited: May 1, 2019]. Available from: https://www.datamation.com/big-data/big-data-visualization.html

[27] WANG, R. Monday's Musings: Beyond The Three V's of Big Data - Viscosity and Virality. *Forbes.* [Online] Feb 27, 2012. [Cited: May 1, 2019]. Available from: https://www.forbes.com/sites/raywang/2012/02/27/mondays-musings-beyond-the-three-vs-of-big-data-viscosity-and-virality

[28] PANIMALAR, Arockia, Varnekha SHREEE and Veneshia KATHERINE. The 17 V's Of Big Data. *International Research Journal of Engineering and Technology.* Sep, 2017, Vol. 7, e-ISSN: 2395-0056.

[29] *KDnuggets.* [Online] [Cited: May 1, 2019]. Available from: https://www.kdnuggets.com/wp-content/uploads/42-vs-2.jpg

[30] Gartner's 2014 Hype Cycle for Emerging Technologies Maps the Journey to Digital Business. *Gartner.* [Online] Aug 11, 2014. [Cited: May 1, 2019]. Available from: https://www.gartner.com/technology/pressRoom.do?id=2819918

[31] DHAR, Vasant. Data science and prediction. *Communications of the ACM.* December 2013, Vol. Issue 12, Volume 56.

[32] LEEK, Jeff. The key word in "Data Science" is not Data, it is Science. *Simply Statistics.* [Online] Dec 12, 2013. [Cited: May 1, 2019]. Available from:

https://simplystatistics.org/2013/12/12/the-key-word-in-data-science-is-not-data-it-is-science/

[33] LESKOVEC, Jure, Anand RAJARAMAN and Jeffrey David ULLMAN. *Mining of Massive Datasets: Second Edition.* Cambridge : Cambridge University Press, 2014. ISBN 978-1-107-07723-2.

[34] DAVENPORT Thomas H., D. J. PATIL. Data Scientist: The Sexiest Job of the 21st Century. *Harward Business Review.* [Online] Oct 2012. [Cited: May 1, 2019]. Available from: https://hbr.org/2012/10/data-scientist-the-sexiest-job-of-the-21st-century.

[35] Data Science. *Wikipedia.* [Online] [Cited: May 1, 2019]. Available from: https://en.wikipedia.org/wiki/Data_science

[36] What is the difference between data science, data analysis, data mining, machine learning, AI, and big data? *Quora.* [Online] Nov 17, 2017. [Cited: May 1, 2019]. Available from: https://www.quora.com/What-is-the-difference-between-data-science-data-analysis-data-mining-machine-learning-AI-and-big-data

[37] ANDERSON, Hugo. What Data Scientists Really Do, According to 35 Data Scientists. *Harward Business Review.* [Online] Aug 15, 2018. [Cited: May 1, 2019]. Available from: https://hbr.org/2018/08/what-data-scientists-really-do-according-to-35-data-scientists

[38] Big Data, Data Analytics, Data Analysis, Data Mining, Data Science & Machine Learning. *Web Scraping.* [Online] Jun 15, 2016. [Cited: May 1, 2019]. Available from: http://scraping.pro/data-analytics-data-analysis-data-mining-data-science-machine-learning-big-data/

[39] Data Science. *Le Big Data.* [Online] [Cited: May 1, 2019]. Available from: https://lebigdata.com/en/data-science/.

[40] TIERNAY, Brendan. Data Science Is Multidisciplinary. *Oralytics.* [Online] Jun 13, 2012. [Cited: May 1, 2019]. Available from: https://www.oralytics.com/2012/06/data-science-is-multidisciplinary.html

[41] Big Data Use Cases. *Big Data Analytics News.* [Online] [Cited: May 1, 2019]. Available from: https://bigdataanalyticsnews.com/big-data-use-cases/

[42] What's driving the connected car. *McKinsey & Company.* [Online] Sep 2014. [Cited: May 1, 2019]. Available from: https://www.mckinsey.com/industries/automotive-and-assembly/our-insights/whats-driving-the-connected-car

[43] Just one autonomous car will use 4,000 GB of data/day. *NetworkWorld from IDG.* [Online] Dec 7, 2016. [Cited: May 1, 2019]. Available from: https://www.networkworld.com/article/3147892/one-autonomous-car-will-use-4000-gb-of-dataday.html

[44] STOJASPAL, Jan. Telematics and the value of Big Data, part I. *TU-Automotive.* [Online] Sep 26, 2013. [Cited: May 1, 2019]. Available from: https://www.tu-auto.com/telematics-and-the-value-of-big-data-part-i/

[45] STOJASPAL, Jan. Telematics and the value of Big Data, part II. *TU-Automotive.* [Online] Oct 3, 2013. [Cited: May 1, 2019]. Available from: https://www.tu-auto.com/telematics-and-the-value-of-big-data-part-ii/

[46] GILBERT, Seth and Nancy LYNCH. Brewer's Conjecture and the Feasibility of Consistent Available Partition-Tolerant Web Services. *ACM SIGACT News.* 2002, 33. 10.1145/564585.564601.

[47] HOLUBOVÁ, Irena, Jiří KOSEK, Karel MINAŘÍK and David NOVÁK. *Big Data a NoSQL databáze.* Prague: Grada, 2015. ISBN 978-80-247-5466-6.

[48] KLEPPMAN, Martin. Please stop calling databases CP or AP. *Martin Kleppmann.* [Online] May 11, 2015. [Cited: May 1, 2019]. Available from: http://martin.kleppmann.com/2015/05/11/please-stop-calling-databases-cp-or-ap.html

[49] MEHRA, Akhil. Understanding the CAP Theorem. *DZone.* [Online] Apr 30, 2019. [Cited: May 1, 2019]. Available from: https://dzone.com/articles/understanding-the-cap-theorem

[50] HURST, Nathan. Visual Guide to NoSQL Systems. *Nathan Hurst's Blog.* [Online] [Cited: May 1, 2019]. Available from: http://blog.nahurst.com/visual-guide-to-nosql-systems

[51] DB-Engines Ranking. *DB-Engines.* [Online] Apr 2019. [Cited: May 1, 2019]. Available from: https://db-engines.com/en/ranking

[52] *Redis.* [Online] [Cited: May 1, 2019]. Available from: http://redis.io

[53] *DynamoDB.* [Online] [Cited: May 1, 2019]. Available from: https://aws.amazon.com/dynamodb/

[54] *Memcached.* [Online] [Cited: May 1, 2019]. Available from: http://www.memcached.org/

[55] *CosmosDB.* [Online] [Cited: May 1, 2019]. Available from: https://azure.microsoft.com/en-us/services/cosmos-db/

[56] *Hazelcast.* [Online] [Cited: May 1, 2019]. Available from: https://hazelcast.com/

[57] *Ehcache.* [Online] [Cited: May 1, 2019]. Available from: http://www.ehcache.org/

[58] *Aeurospike.* [Online] [Cited: May 1, 2019]. Available from: https://www.aerospike.com/

[59] *OrientDB.* [Online] [Cited: May 1, 2019]. Available from: https://orientdb.com/

[60] *Riak KV.* [Online] [Cited: May 1, 2019]. Available from: https://riak.com/products/riak-kv/

[61] *Apache Ignite.* [Online] [Cited: May 1, 2019]. Available from: https://ignite.apache.org/

[62] *MongoDB.* [Online] [Cited: May 1, 2019]. Available from: https://www.mongodb.com/

[63] *Couchbase.* [Online] [Cited: May 1, 2019]. Available from: https://www.couchbase.com/

[64] *CouchDB.* [Online] [Cited: May 1, 2019]. Available from: http://couchdb.apache.org/

[65] *MarkLogic.* [Online] [Cited: May 1, 2019]. Available from: https://www.marklogic.com/

[66] *Firebase.* [Online] [Cited: May 1, 2019]. Available from: https://firebase.google.com/products/realtime-database/

[67] *RavenDB.* [Online] [Cited: May 1, 2019]. Available from: https://ravendb.net/

[68] Google Cloud. *Cloud Datastore.* [Online] [Cited: May 1, 2019]. Available from: https://cloud.google.com/datastore/

[69] *Apache Cassandra.* [Online] [Cited: May 1, 2019]. Available from: http://cassandra.apache.org/

[70] *Apache HBase.* [Online] [Cited: May 1, 2019]. Available from: http://hbase.apache.org/

[71] *Datastax.* [Online] [Cited: May 1, 2019]. Available from: https://www.datastax.com/products/datastax-enterprise

[72] *Microsoft Azure Table Storage.* [Online] [Cited: May 1, 2019]. Available from: https://azure.microsoft.com/en-us/services/storage/tables/

[73] *Apache Accumulo.* [Online] [Cited: May 1, 2019]. Available from: https://accumulo.apache.org/

[74] Google Cloud. *Cloud Bigtable.* [Online] [Cited: May 1, 2019]. Available from: https://cloud.google.com/bigtable/

[75] *ScyllaDB.* [Online] [Cited: May 1, 2019]. Available from: https://www.scylladb.com/

[76] *MapR DB.* [Online] [Cited: May 1, 2019]. Available from: https://mapr.com/products/mapr-db/

[77] *Alibaba Cloud Table.* [Online] [Cited: May 1, 2019]. Available from: https://www.alibabacloud.com/product/table-store

[78] *Neo4J.* [Online] [Cited: May 1, 2019]. Available from: https://neo4j.com/

[79] *ArangoDB.* [Online] [Cited: May 1, 2019]. Available from: https://www.arangodb.com/

[80] *OpenLink Virtuoso.* [Online] [Cited: May 1, 2019]. Available from: https://virtuoso.openlinksw.com/

[81] *Amazon Neptun.* [Online] [Cited: May 1, 2019.] Available from: https://aws.amazon.com/neptune/.

[82] *JanusGraph.* [Online] [Cited: May 1, 2019.] Available from: https://janusgraph.org/

[83] *Apache Giraph.* [Online] [Cited: May 1, 2019]. Available from: http://giraph.apache.org/

[84] *Dgraph.* [Online] [Cited: May 1, 2019]. Available from: https://dgraph.io/

[85] *GraphDB.* [Online] [Cited: May 1, 2019]. Available from: http://graphdb.ontotext.com/

[86] Multi-model database. *Wikipedia.* [Online] [Cited: May 1, 2019]. Available from: https://en.wikipedia.org/wiki/Multi-model_database

[87] Hadoop Ecosystem and Their Components. *Data-Flair.* [Online] [Cited: May 1, 2019]. Available from: https://data-flair.training

[88] GHEMAWAT, Sanjay, Howard GOBIOFF and Shun-Tak LEUNG. The Google File System. *Google Research.* [Online] [Cited: May 1, 2019]. Available from: https://research.google.com/archive/gfs-sosp2003.pdf

[89] DEAN, Jeffrey and Sanjay GHEMAVAT. MapReduce: Simplified Data Processing on Large Clusters. *Google Research.* [Online] [Cited: May 1, 2019]. Available from: https://research.google.com/archive/mapreduce-osdi04.pdf

[90] *Apache Hadoop.* [Online] [Cited: May 1, 2019]. Available from: https://hadoop.apache.org/

[91] AVEN, Jeffrey. *Sams teach yourself hadoop in 24 hours.* Old Tappan, NJ: Pearson Education, 2017. ISBN 978-0-672-33852-6.

[92] *Apache Projects.* [Online] [Cited: May 1, 2019]. Available from: https://projects.apache.org/projects.html?category

[93] Apache Hadoop. *Wikipedia.* [Online] [Cited: May 1, 2019]. Available from: https://en.wikipedia.org/wiki/Apache_Hadoop

[94] Apache Hadoop Release Versioning. *Apache Hadoop.* [Online] [Cited: May 1, 2019]. Available from: https://hadoop.apache.org/versioning.html

[95] MONE, Gregory. Beyond Hadoop. *Communications of the ACM.* 2013, Vol. Vol. 56, No. 1.

[96] *Apache Tez.* [Online] [Cited: May 1, 2019]. Available from: https://tez.apache.org/

[97] Hadoop Common. *Techopedia.* [Online] [Cited: May 1, 2019]. Available from: https://www.techopedia.com/definition/30427/hadoop-common

[98] The Small Files Problem. *Cloudera.* [Online] Feb 2, 2009. [Cited: May 1, 2019]. Available from: https://blog.cloudera.com/blog/2009/02/the-small-files-problem/

[99] Data Write Operation in HDFS. *Core Java Guru.* [Online] [Cited: May 1, 2019]. Available from: http://www.corejavaguru.com/bigdata/hadoop/hdfs-file-write

[100] Hadoop Data Write Operation Acknowledgement. *Stack Overflow.* [Online] [Cited: May 1, 2019]. Available from: https://stackoverflow.com/questions/32038000/hadoop-2-0-data-write-operation-acknowledgement

[101] Hadoop HDFS Data Read and Write Operations. *Data Flair Training.* [Online] Nov 14, 2018. [Cited: May 1, 2019]. Available from: https://data-flair.training/blogs/hadoop-hdfs-data-read-and-write-operations/

[102] Apache Hadoop YARN – Concepts and Applications. *Hortonworks.* [Online] [Cited: May 1, 2019]. Available from: https://hortonworks.com/blog/apache-hadoop-yarn-concepts-and-applications/

[103] Hadoop MapReduce Comprehensive Description. *Distributed Systems Architecture.* [Online] [Cited: May 1, 2019]. Available from: https://0x0fff.com/hadoop-mapreduce-comprehensive-description/

[104] BAKSHI, Ashish. MapReduce Tutorial – Fundamentals of MapReduce with MapReduce Example. *Edureka.* [Online] Dec 25, 2018. [Cited: May 1, 2019]. Available from: https://www.edureka.co/blog/mapreduce-tutorial/

[105] *Apache Spark.* [Online] [Cited: May 1, 2019]. Avalaible from: https://spark.apache.org/

[106] KARAU, Holden, Andy KONWINSKI, Patrick WENDELL and Matei ZAHARIA. *Learning Spark.* Sebastopol: O'Reilly, 2015. ISBN 978-1-449-35862-4.

[107] LYNN, Dan. Apache Spark Cluster Managers: YARN, Mesos, or Standalone? *AgilData.* [Online] Mar 15, 2016. [Cited: May 1, 2019]. Available from: http://www.agildata.com/apache-spark-cluster-managers-yarn-mesos-or-standalone/

[108] Spark SQL, DataFrames and Datasets Guide. *Apache Spark.* [Online] [Cited: May 1, 2019]. Available from: https://spark.apache.org/docs/latest/sql-programming-guide.html

LIST OF ABBREVIATIONS

ACID	**A**tomicity, **C**onsistency, **I**solation, **D**urability
BASE	**B**asic **A**vailability, **S**oft state, **E**ventual consistency
BSON	**B**inary J**SON**
CAP	**C**onsistency, **A**vailability, **P**artition tolerance
CRUD	**C**reate, **R**ead, **U**pdate, **D**elete
DBMS	**D**atabase **M**anagement **S**ystem
HDFS	**H**adoop **D**istributed **F**ile **S**ystem
HQL	**H**ive S**QL**
IoT	**I**nternet **o**f **T**hings
JSON	**J**avaScript **O**bject **N**otation
NoSQL	**N**ot **O**nly **SQL**
OLAP	**O**nline **A**nalytical **P**rocessing
OLTP	**O**nline **T**ransaction **P**rocessing
RDBMS	**R**elational **D**atabase **M**anagement **S**ystem
SQL	**S**tructure **Q**uery **L**anguage
SSH	**S**ecure **Sh**ell
XML	e**X**tensible **M**arkup **L**anguage
YARN	**Y**et **A**nother **R**esource **N**egotiator

LIST OF FIGURES

LIST OF TABLES

LIST OF SOFTWARE

Software	Version	Released on
OS Ubuntu Server	18.04.2 LTS Bionic	Feb 14, 2019
Java 8 JDK	1.8.212	Apr 16, 2019
Apache Hadoop	3.1.2	Feb 6, 2019
Apache Spark	2.4.2	Apr 23, 2019
Apache Pig	0.17.0	Jun 19, 2017
Apache Hive	3.1.1	Nov 1, 2018
Apache Sqoop	1.4.7	Jan 24, 2018
MySQL	5.7.26	Apr 25, 2019