MODULE I

1

INTRODUCTION TO BIG DATA AND HADOOP

Unit Structure

1.0 OBJECTIVES

After going through this unit, you will be able to:

- Define data analytics on huge data sets
- Extract meaningful insights, such as hidden patterns and unknown correlations.
- Explain the steps in data pre-processing
- describe the dimensionality of data
- learn the data reduction and data compression techniques

1.1 INTRODUCTION TO BIGDATA

Big Data is quickly becoming one of the most discussed technological phenomena today. The true issue for large organizations is to make the most of the data that is already available and foresee what type of problems may arise? What data is to be gathered in the future? How can we take the existing data and make it relevant for us? Accurate insight into prior data is a crucial discussion point in many executive meetings.

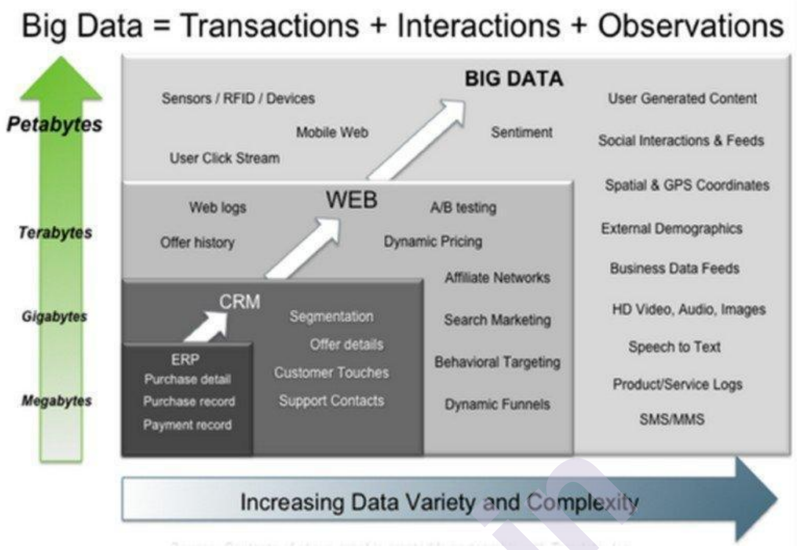


Figure 1.1: Bigdata - Transactions, interactions, and Observations

Within organizations with the expansion of data, the challenge has risen to new heights, and Big Data is now a reality. In many companies, this is the case. The objective of every company and expert is the same: to maximize profits. The path and starting point are derived from the data. Organizations are learning about the approaches and possibilities associated with Big Data as they evaluate and develop big data solutions.

There is no one answer to big data, and no single provider can claim to know everything about big data. Big Data is an overly broad notion with several actors - various architectures, providers, and technologies. The three Vs of Big data are Velocity, Volume and Variety shown in figure 1.2

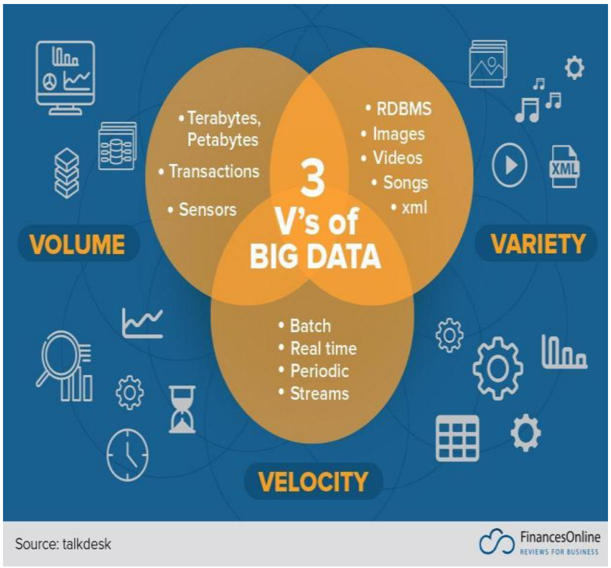


Figure 1.2: V's of Bigdata

1.2 BIGDATA CHARACTERISTICS

Gartner analyst Doug Laney identified the three 'Vs of Big Data in 2001: Variety, Velocity, and Volume. Let us now look at the properties of large data. These properties alone are sufficient to understand what big data is. Let's take a closer look at them:

Volume:

The exponential rise of data storage when data becomes more than just text data. On our social media platforms, the data is available in the form of films, music, and enormous photos. Terabytes and Petabytes of storage are increasingly popular in business storage systems. As the database increases in size, the applications and architecture designed to support it must be re-evaluated on a regular basis.

When the same data is re-evaluated from numerous perspectives, even though the original data remains the same, the new uncovered intelligence causes an explosion of the data. The large amount does, in fact, reflect Big Data.

Velocity:

The rise of data and social media has altered our perspective on data. There was a time when we thought data from yesterday was current. In reality, newspapers continue to follow that rationale. However, news networks and radios have transformed how quickly we acquire information.

People nowadays respond on social media to keep them up to speed on what is going on. A few seconds' old messages (tweets, status updates, etc.) on social media may not be of interest to users.

They often delete old messages and focus on fresh changes. Data transfer is now nearly real-time, and the update window has shrunk to fractions of a second. Big Data is represented by this high-velocity data.

Variety:

Data may be stored in a variety of formats. For example, it may be saved in a database, excel, csv, access, or even a plain text file. Sometimes the data is not even in the typical format that we expect; it may be in the form of video, SMS, pdf, or something else that we haven't considered. It is the organization's responsibility to organize it and make it relevant. It will be simple if we have data in the same format, but this is not always the case. The actual world has data in a variety of forms, which is the problem we must conquer with Big Data. This variety of data represents Bigdata.

1.3 TYPES OF BIGDATA

Every day, people create 2.5 quintillion bytes of data. According to Statista, the internet will create 74 Zettabytes (74 trillion GBs) of data by the end of 2021. Managing such a massive and ongoing data outsourcing is becoming increasingly tough. So, in order to manage such massively complicated data, big data was established; it is concerned with the transformation of vast and complex data into useful data that cannot be collected or analyzed using standard methods.

It is impossible to save all data in the same way. After the type of data has been recognized, the techniques for data storage may be correctly examined. A Cloud Service, such as Microsoft Azure, is a one-stop-shop for storing all types of data; blobs, queues, files, and so on.

Azure Cloud Services such as Azure SQL and Azure Cosmos DB, for example, aid with the handling and management of sparsely diverse types of data.

1.3.1 Structured:

Structured data is one sort of large data. Structured data is defined as data that can be processed, stored, and retrieved in a consistent way. It refers to information that is neatly structured and can be easily and smoothly stored and accessed from a database using basic search engine methods. For example, the employee table in a corporate database will be formatted such that personnel details, work positions, salary, and so on are all presented in an ordered manner.

Example: An Employee table in a database is an example of Structured Data

Emp_ID	Employee_Name	Gender	Department	Salary
2265	Rajesh	Male	Admin	60000
3547	Prathiba Joshi	Female	Finance	75000
7214	Shushil Roy	Male	Admin	50000
7499	Priya Sane	Female	HR	80000

1.3.2 Unstructured:

Unstructured data is data that does not have any defined shape or organization. This makes processing and analyzing unstructured data extremely complex and time-consuming. Unstructured data is something like email. Big data may be classified into two types: structured and unstructured. Today's enterprises have a lot of data at their disposal, but they don't know how to extract value from it since the data is in its raw form or unstructured format.

Example: The output returned by 'Google Search'

1.3.3 Semi-Structured:

The third category of big data is semi-structured data. Semi-structured data is data that has both the above-mentioned types, namely structured and unstructured data. To be more specific, it refers to data that, while not categorized under a certain repository (database), contains critical information or tags that separate different pieces within the data. Thus, we have reached the end of the data kinds.

Example of Semi-structured data: A data represented in an XML File:

```
<rec>
<name> Kishore Rao</name>
<sex> Male</sex>
<age> 35</age>
</rec>
```

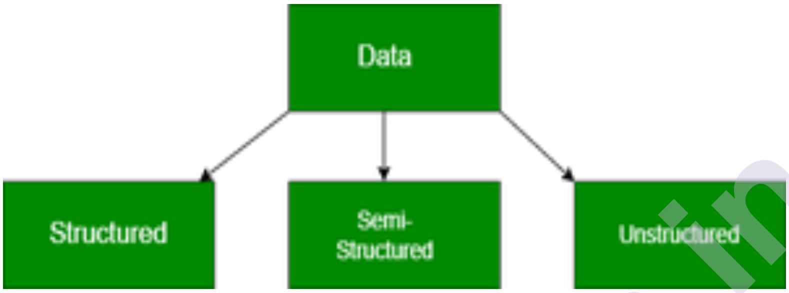


Figure 1.3 Types of Bigdata

1.4 TRADITIONAL VS BIGDATA

Big data offers enormous benefits for organizations, such as deeper insights into consumer behavior, more accurate projections of market activity, and overall enhanced efficiency. Every year, people and companies generate an increasing amount of data. In 2010, the globe generated only 1.2 zettabytes (1.2 trillion gigabytes) of fresh data, according to an IDC analysis. It might reach 175 zettabytes (175 trillion gigabytes) or more by 20251. As organizations use predictive analytics and data mining to tap into this thriving resource, the market for big data will expand as well. According to Statista, the big data market will more than double from $169 billion to $274 billion between 2018 and 2027.

What, then, are the major distinctions between big data and traditional data? And how do they affect present data storage, processing, and analysis technology? Here, we'll describe the many functions of each form of data while highlighting the need for a strategy that considers both big data and traditional data.

1.4.1 What Is A Conventional System:

Conventional systems are made up of one or more zones, each with either humanly controlled call points or automatic detection equipment, or a combination of both. Big data is a massive volume of data that exceeds the processing capabilities of conventional systems. Conventional data is organized, and companies have been storing and analyzing relational data for decades. The majority of the world's data is still conventional data.

Traditional data may be used by businesses to measure sales, manage client relationships, and manage workflows. Traditional data is frequently easier to alter and may be managed using standard data processing applications. However, compared to big data, it often gives fewer complex insights and more restricted advantages.

1.4.2 Difference Between Conventional Computing And Intelligent Computing:

- Traditional computing works logically with a set of rules and computations, whereas neural computing may work with visuals, pictures, and concepts.
- Conventional computing is frequently incapable of dealing with the unpredictability of data acquired in the actual world.
- On the other hand, neural computing, like our own brains, is ideally adapted to problems with no apparent algorithmic answers and is capable of dealing with noisy imprecise input. This enables them to succeed in areas where traditional computing typically fails.
- Traditional computing works logically with a set of rules and computations, whereas Intelligent computing may work with visuals, pictures, and concepts.
- Traditional computing is frequently incapable of dealing with the unpredictability of data acquired in the real world. On the other hand, computing, like our own brains, is best adapted to problems with no apparent algorithmic answers and is capable of dealing with noisy, inaccurate input. This enables them to succeed in areas where traditional computing typically fails.

1.4.3 Comparison of Bigdata with Conventional Data:

Big Data	Conventional Data
Huge data sets	Data set size in control
Unstructured data such as text, video, and audio.	Normally structured data such as numbers and categories, but it can take other forms as well.
Hard-to-perform queries and analysis	Relatively easy-to-perform queries and analysis.
Needs a new methodology for analysis.	Data analysis can be achieved by using conventional methods
Need tools such as Hadoop, Hive, Hbase, Pig, Sqoop, and so on.	Tools such as SQL, SAS, R, and Excel alone may be sufficient
The aggregated or sampled or filtered data.	Raw transactional data.
Used for reporting, basic analysis, and text mining. Advanced analytics is only in the starting stage in big data.	Used for reporting, advanced analysis, and predictive modeling.

Big data analysis needs both programming skills (such as Java) and analytical skills to perform analysis.	Analytical skills are sufficient for conventional data; advanced analysis tools don't require expert programming skills.
Petabytes/exabytes of data	Millions/billions of accounts.
Billions/trillions of transactions.	Megabytes/gigabytes of data
Thousands/millions of accounts.	Millions of transactions
Generated by big financial institutions, Facebook, Google, Amazon, eBay, Walmart, and so on.	Generated by small enterprises and small banks.

1.4.4 Challenges of Conventional Systems:

In the case of Conventional systems in real-time applications, the following issues have predominated:

1) Data Management Landscape Uncertainty
2) The Talent Gap in Big Data
3) The industry's existing talent gap Data entry into the big data platform
4) The requirement for data source synchronization
5) Obtaining critical insights through the application of big data analytics

1.4.5 Challenges of Bigdata:

Big Data Challenges include the best approach to handle a large amount of data, which includes the process of storing and analyzing a large collection of data on multiple data repositories. There are several main issues that arise while working with Big Data that must be addressed with agility.

- **Lack of professional's Knowledge:**

Companies want competent data specialists to run the latest technology and huge Data tools. To work with the technologies and make sense of massive data sets, these experts will include data scientists, data analysts, and data engineers. One of the most difficult Big Data challenges that any company faces is the scarcity of enormous Data specialists. This is frequently due to the fact that data handling tools have advanced fast, but most experts have not. To close this gap, concrete efforts must be done.

- **Lack of proper understanding of massive data:**

Companies fail in their Big Data endeavors due to a lack of knowledge. Employees may be unaware of what data is, how it is stored, and processed, how important it is, and where it comes from. While data specialists may be aware of what is going on, others may not have a clear picture. For example, if staff does not appreciate the necessity of knowledge storage, they may fail to retain critical data backups. They were unable to correctly save data in databases. As a result, when this critical information is needed, it is difficult to obtain.

- **Data Growth Issues:**

One of the most important difficulties of Big Data is appropriately storing these vast amounts of knowledge. The amount of knowledge held in corporate data centres and databases is continually expanding. It becomes difficult to manage large data sets as they increase rapidly over time. The majority of the data is unstructured and derived from documents, movies, audio files, text files, and other sources. This implies that they are not in the database.

- **Confusion While Selecting Bigdata Tool:**

Companies are frequently perplexed while deciding on the simplest tool for giants. Analysis and storage of data Is HBase or Cassandra the most basic data storage technology? Is Hadoop MapReduce sufficient, or will Spark be a far superior choice for data analytics and storage? These inquiries irritate businesses, and they are sometimes unable to find solutions. They end up making bad judgments and using incorrect technology. As a result, money, time, effort, and working hours are all squandered.

- **Integrating data from a spread of sources:**

In an organization, data comes from a variety of sources, including social media sites, ERP software, customer logs, financial reports, e-mails, presentations, and employee-created reports. Combining all of this data to arrange reports might be a difficult undertaking. This is a neighborhood that many businesses overlook. It's ideal since data integration is critical for analysis, reporting, and business intelligence.

- **Securing Data:**

One of the challenging issues of enormous Data is securing these vast volumes of knowledge. Companies are frequently so preoccupied with understanding, storing, and analyzing their data sets that they postpone data security until later stages. This is not always a wise decision because unsecured data repositories may become breeding grounds for malevolent hackers. A stolen record or a knowledge breach may cost a company up to $3.7 million.

1.5 BIGDATA APPLICATIONS

In today's world, there are a lot of data. Big companies utilize those data for their business growth. By analyzing this data, the useful decision can be made in various cases as discussed below:

- **Tracking Customer Spending Habit, Shopping Behavior:**

In large retail stores (such as Amazon, Walmart, and Big Bazar), the management team must retain data on client spending habits (which product the customer spent, which band they intend to spend, and how frequently they spent), purchasing behavior, and the customer's favorite product (so that they can keep those products in the store). Based on whatever product is being searched / sold the most, the production / collection rate of that product is fixed. The banking sector uses its customers' spending behavior-related data to present an offer to a specific consumer to buy his or her desired goods using the bank's credit or debit card with a discount or reward. They may send the right offer to the right individual at the right time this way.

- **Recommendation:**

Big retail stores make recommendations to customers by analyzing their spending habits and purchasing activity. Product recommendations are made by e-commerce sites such as Amazon, Walmart, and Flipkart. They track what product a consumer is looking for and then propose that product to that customer based on that data. Assume a consumer searches for a bed cover on Amazon. As a result, Amazon learned that a client could be interested in purchasing a bed cover. The next time the consumer visits a Google website, he or she will see advertisements for various bed coverings. As a result, the proper product may be advertised to the right buyer. YouTube also suggests videos based on the user's prior liked and viewed video types. During video playback, appropriate advertisements are provided based on the content of the video the viewer is watching. Assume someone is viewing a Big Data lesson video, and an advertisement for another Big Data course is displayed during that video.

- **Smart Traffic System:**

Data on the status of traffic on various roads were obtained using a camera mounted beside the road, at the city's entry and departure points, and a GPS device installed in the car (Ola, Uber cab, etc.). All of this data is examined, and jam-free or less jam-prone routes that take less time are advised. Big data analysis may be used to build a smart traffic system in the city in this manner. Another advantage is that fuel usage may be lowered.

- **Virtual Personal Assistant Tool:**

Big data analysis enables virtual personal assistant tools (such as Siri on Apple devices, Cortana on Windows, and Google Assistant in Android) to respond to numerous questions posed by users. This program monitors the user's location, local time, season, and other data connected to the query asked, among other things. It delivers a response after analyzing all such data.

As an example, assume a user asks, "Do I need to bring an umbrella?" The tool collects data such as the user's location, season, and weather condition in that area then analyses this data to determine if there is a probability of rain and then provides the response.

- **IoT:**

Machines with IoT sensors are installed by manufacturing companies to collect operational data. By analyzing such data, it is possible to anticipate how long a machine will run without issue until it has to be repaired, allowing the firm to take action before the equipment encounters a number of problems or fails completely. As a result, the cost of replacing the entire equipment can be avoided.

Big data is making a huge impact in the realm of healthcare. Data on patient experiences is collected using a big data platform and used by clinicians to provide better care. An IoT gadget can detect a sign of a potentially fatal disease in the human body and prevent it from providing early treatment.

An IoT sensor put near a patient or a newborn infant continually monitors different health conditions such as heart rate, blood pressure, and so on. When any parameter exceeds the safe limit, an alarm is transmitted to a doctor, allowing them to take action remotely as soon as possible.

1.6 LET US SUM UP

- Big data is a field that treats ways to analyze, systematically extract information from or otherwise deal with data sets that are too large or complex to be dealt with by traditional data-processing application software.
- Dubbed the three Vs; volume, velocity, and variety, these are key to understanding how we can measure big data and just how very different 'big data is from old-fashioned data.
- Big data is a combination of structured, semi-structured, and unstructured data collected by organizations that can be mined for information and used in machine learning projects, predictive modeling, and other advanced analytics applications.
- Big data makes it possible for you to gain more complete answers because you have more information.
- More complete answers mean more confidence in the data—which means a completely different approach to tackling problems.

1.7 LIST OF REFERENCES

1. Tom White, “HADOOP: The definitive Guide” O Reilly 2012, Third Edition, ISBN: 978-1-449-31152-0
2. Chuck Lam, “Hadoop in Action”, Dreamtech Press 2016, First Edition, ISBN:13 9788177228137
3. Shiva Achari,” Hadoop Essential “PACKT Publications, ISBN 978-1-78439- 668-8.

4. RadhaShankarmani and M. Vijayalakshmi," Big Data Analytics "Wiley Textbook Series, Second Edition, ISBN 9788126565757.

5. Jeffrey Aven," Apache Spark in 24 Hours" Sam's Publication, First Edition, ISBN: 0672338513S.

6. Bill Chambers and MateiZaharia," Spark: The Definitive Guide: Big Data Processing Made Simple "O'Reilly Media; First edition, ISBN-10: 1491912219;

7. James D. Miller," Big Data Visualization" PACKT Publications. ISBN-10: 1785281941.

8. Judith Huruwitz, Alan Nugent, Fern Halper, Marcia Kaufman, "Big data for dummies", John Wiley & Sons, Inc. (2013)

9. Tom White, "Hadoop the Definitive Guide", O'Reilly Publications, Fourth Edition,2015

10. Dirk Deroos, Paul C. Zikopoulos, Roman B. Melnky, Bruce Brown, Rafael Coss, "Hadoop for Dummies", Wiley Publications,2014

11. Robert D. Schneider, "Hadoop for Dummies", John Wiley & Sons, Inc. (2012)

12. Paul Zikopoulos, "Understanding Big Data: Analytics for Enterprise Class Hadoop and Streaming Data, McGraw Hill, 2012 Chuck Lam, "Hadoop in Action", Dreamtech Publications, 2010.

1.8 WEB REFERENCES

1. https://hadoop.apache.org/docs/stable/

2. https://pig.apache.org/

3. https://hive.apache.org/

4. https://spark.apache.org/documentation.html

5. https://help.tableau.com/current/pro/desktop/en-us/default.htm

1.9 UNIT END EXERCISES

1. What is Bigdata?

2. List out the best practices of big data Analytics.

3. Write down the characteristics of big data Applications.

4. Name the computing resources of big data storage.

5. List out some big data platforms.

*****

2

INTRODUCTION TO BIG DATA AND HADOOP

Unit Structure

2.0 OBJECTIVES

After going through this unit, you will be able to:

- Define data analytics on huge data sets
- Extract meaningful insights, such as hidden patterns and unknown correlations.
- Explain the steps in data pre-processing
- describe the dimensionality of data
- learn the data reduction and data compression techniques

2.1 INTRODUCTION TO HADOOP

Hadoop is a Java-based Apache open-source framework that allows for the distributed processing of large datasets across clusters of computers using simple programming models. The Hadoop framework application operates in a computing environment that allows for distributed storage and computation across computer clusters. Hadoop is intended to scale from a single server to thousands of machines, each of which provides local computation and storage.

Why use Hadoop:

- Apache Hadoop is not only a storage system but also data storage and processing platform.
- It is expandable (as we can add more nodes on the fly).
- Tolerant to faults (Even if nodes go down, data is processed by another node).
- It efficiently processes large amounts of data on commodity hardware in a cluster.
- Hadoop is used to process massive amounts of data.
- Commodity hardware is low-end hardware; it consists of inexpensive devices that are very cost-effective. As a result, Hadoop is very cost-effective.

Hadoop modules:

- **HDFS:** Hadoop Distributed File System Google released its GFS paper, and HDFS was built on top of it. It specifies that the files will be divided into blocks and stored in distributed architecture nodes.
- **Yarn:** Yet another Resource Negotiator that is used for job scheduling and cluster management.
- **Map Reduce:** is a framework that allows Java programmes to perform parallel computations on data using key-value pairs. The Map task converts input data into a data set that can be computed in Key value pair. The output of the Map task is consumed by the Reduce task, and the output of the Reducer produces the desired result.
- **Hadoop common:** These Java libraries are used to get things started.

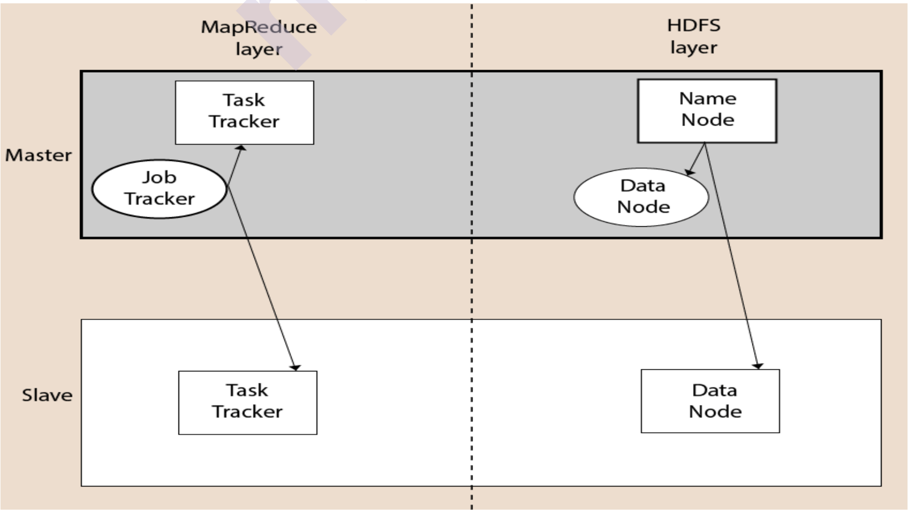


Figure 2.1 Framework of Hadoop

2.2 HISTORY OF HADOOP

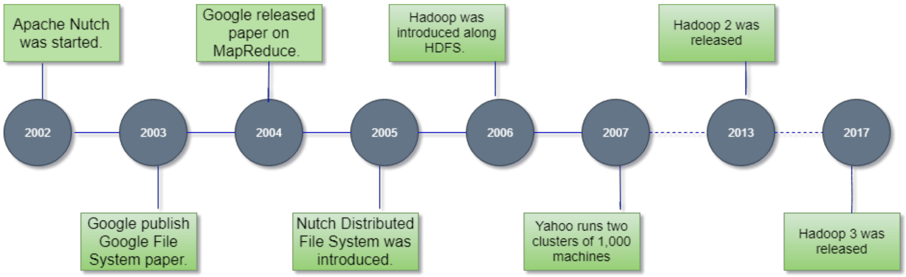

Apache Nutch was started.
Google released paper on MapReduce.
Hadoop was introduced along HDFS.
Hadoop 2 was released
2002
2003
2004
2005
2006
2007
2013
2017
Google publish Google File System paper.
Nutch Distributed File System was introduced.
Yahoo runs two clusters of 1,000 machines
Hadoop 3 was released

Figure 2.2 History of Hadoop

- Doug Cutting and Mike Cafarella founded Hadoop in 2002. Its origins can be traced back to the Google File System paper, which was published by Google. Let's focus on the history of Hadoop in the following steps:

- Doug Cutting and Mike Cafarella began working on Apache Nutch in 2002. It is a web crawler software project that is open source.

- They were dealing with large amounts of data while working on Apache Nutch. They have to spend a lot of money to store that data, which is the result of that project. This problem becomes one of the primary reasons for the development of Hadoop.

- Google introduced the GFS file system in 2003. (Google file system). It is a proprietary distributed file system designed to provide fast data access.

- Google published a white paper on Map Reduce in 2004. This method streamlines data processing on large clusters.

- Doug Cutting and Mike Cafarella introduced the NDFS file system in 2005. (Nutch Distributed File System). Map reduce is also included in this file system.

- Doug Cutting left Google in 2006 to join Yahoo. Dough Cutting introduces a new project Hadoop with a file system known as HDFS based on the Nutch project (Hadoop Distributed File System). This year saw the release of Hadoop's first version, 0.1.0.

- Doug Cutting named his Hadoop project after his son's toy elephant.

- Yahoo operates two 1000-machine clusters in 2007.

- Hadoop became the fastest system in 2008, sorting 1 terabyte of data on a 900-node cluster in 209 seconds.

- Hadoop 2.2 was released in 2013.

- Hadoop 3.0 was released in 2017.

2.3 HADOOP ARCHITECTURE

The Hadoop architecture consists of the file system, the MapReduce engine, and the HDFS (Hadoop Distributed File System). MapReduce engines are either MapReduce / MR1 or YARN / MR2.

A Hadoop cluster is made up of a single master and several slave nodes. Job Tracker, Task Tracker, Name Node, and Data Node comprise the master node, while Data Node and Task Tracker comprise the slave node.

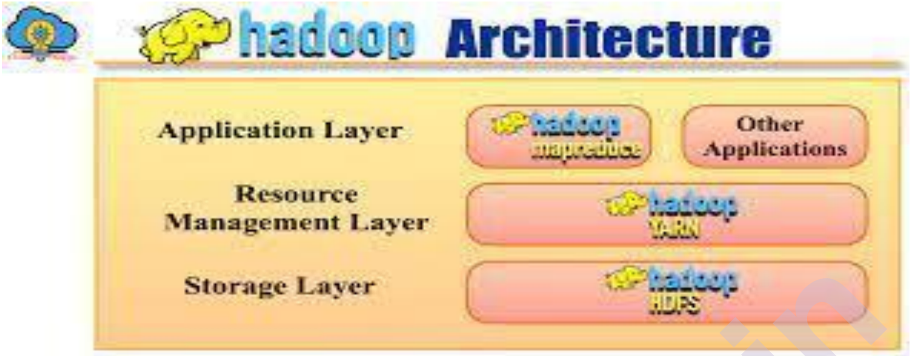


Figure 2.3 Hadoop Architecture

2.4 HADOOP DISTRIBUTED FILE SYSTEM

The Hadoop Distributed File System (HDFS) is a Hadoop distributed file system. It is built with master/slave architecture. This architecture consists of a single Name Node acting as the master and multiple Data Nodes acting as slaves.

Name Node and Data Node are both capable of running on commodity machines. HDFS is written in the Java programming language. As a result, the Name Node and Data Node software can be run on any machine that supports the Java programming language.

The Hadoop Distributed File System (HDFS) is a distributed file system based on the Google File System (GFS) that is designed to run on commodity hardware. It is very similar to existing distributed file systems. However, there are significant differences between it and other distributed file systems. It is highly fault-tolerant and is intended for use on low-cost hardware. It allows for high-throughput access to application data and is appropriate for applications with large datasets. In the HDFS cluster, there is only one master server.

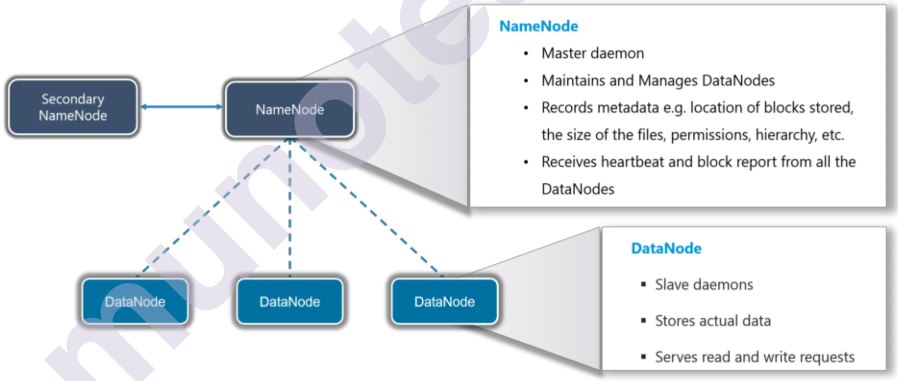
- Because it is a single node, it may be the source of a single point failure.
- It manages the file system namespace by performing operations such as file opening, renaming, and closing.
- It simplifies the system's architecture.

Name Node:

- In the HDFS cluster, there is only one master server.
- Because it is a single node, it may be the source of a single point failure.
- It manages the file system namespace by performing operations such as file opening, renaming, and closing.
- It simplifies the system's architecture.

Data Node:

- There are several Data Nodes in the HDFS cluster.
- Each Data Node contains a number of data blocks.
- These data blocks are used to save information.
- It is Data Node's responsibility to handle read and write requests from file system clients.
- When instructed by the Name Node, it creates, deletes, and replicates blocks.



NameNode

- Master daemon
- Maintains and Manages DataNodes
- Records metadata e.g. location of blocks stored, the size of the files, permissions, hierarchy, etc.
- Receives heartbeat and block report from all the DataNodes

DataNode

- Slave daemons
- Stores actual data
- Serves read and write requests

Figure 2.4: HDFS

Job Tracker:

- Job Tracker's role is to accept MapReduce jobs from clients and process the data using Name Node.
- Name Node responds by sending metadata to Job Tracker.

Task Tracker:

- It performs the function of a slave node for Job Tracker.
- It receives the task and code from Job Tracker and applies it to the file. This procedure is also known as a Mapper.

Mapreduce Layer:

When the client application submits the MapReduce job to Job Tracker, the MapReduce is created. As a result, the Job Tracker forwards the request to the relevant Task Trackers. The Task Tracker occasionally fails or times out. That portion of the job is rescheduled in such a case.

2.4.1 Advantages and Disadvantages of HDFS:

Advantages of HDFS:

The advantages include its low cost, immutability, ability to store data consistently, tolerability of errors, scalability, block structure, capacity to handle massive amounts of data concurrently, and many others.

Disadvantages of HDFS:

The main problem is that it is not suitable for little amounts of data. It also has difficulties with possible stability, as well as being restricting and abrasive in character.

Apache Flumes, Apache Oozie, Apache HBase, Apache Sqoop, Apache Spark, Apache Storm, Apache Pig, Apache Hive, Apache Phoenix, and Cloudera Impala are also supported.

2.4.2 How Does Hadoop Work?

It is quite expensive to build larger servers with heavy configurations that handle large-scale processing, but as an alternative, you can connect many commodity computers with single-CPU as a single functional distributed system, and the clustered machines can read the dataset in parallel and provide much higher throughput. Furthermore, it is less expensive than purchasing a single high-end server. So, the fact that Hadoop runs on clustered and low-cost machines is the first motivator for using it.

- Initially, data is organized into directories and files.
- The files are divided into 128M and 64M uniformly sized blocks (preferably 128M).
- The processing is overseen by HDFS, which sits on top of the local file system.
- To handle hardware failure, blocks are replicated.
- Checking to see if the code was successfully executed.
- Executing the sort that occurs between the maps and reduce stages.
- Sending sorted data to a specific computer.
- For each job, create debugging logs.

2.4.3 Where is Hadoop Used?

Hadoop is used for:

- Search – Yahoo, Amazon, Zvents
- Log processing – Facebook, Yahoo
- Data Warehouse – Facebook, AOL
- Video and Image Analysis – New York Times, Eyealike

Till now, we have seen how Hadoop has made Big Data handling possible. But there are some scenarios where Hadoop implementation is not recommended.

When not to use Hadoop?

Following are some of those scenarios:

- Low Latency data access: Quick access to small parts of data
- Multiple data modification: Hadoop is a better fit only if we are primarily concerned about reading data and not modifying data.
- Lots of small files: Hadoop is suitable for scenarios, where we have few but large files.

After knowing the best suitable use-cases, let us move on and look at a case study where Hadoop has done wonders.

2.5 OVERVIEW OF YARN

Hadoop's resource management layer is Apache Yarn, which stands for "***Yet Another Resource Negotiator."*** The yarn was first introduced in Hadoop 2. x. Yarn enables various data processing engines, including as graph processing, interactive processing, stream processing, and batch processing, to operate and handle HDFS data (Hadoop Distributed File System). Yarn offers work schedule in addition to resource management.

Yarn extends Hadoop's capabilities to other developing technologies, allowing them to benefit from HDFS (the most stable and popular storage system on the globe) and economic cluster.

Apache yarn is also a Hadoop 2.x data operating system. This Hadoop 2.x architecture provides a general-purpose data processing platform that is not confined to MapReduce.

It enables Hadoop to handle systems other than MapReduce that are purpose-built for data processing. It enables the execution of many frameworks on the same hardware as Hadoop.

The Figure below demonstrates the architecture of Hadoop YARN.

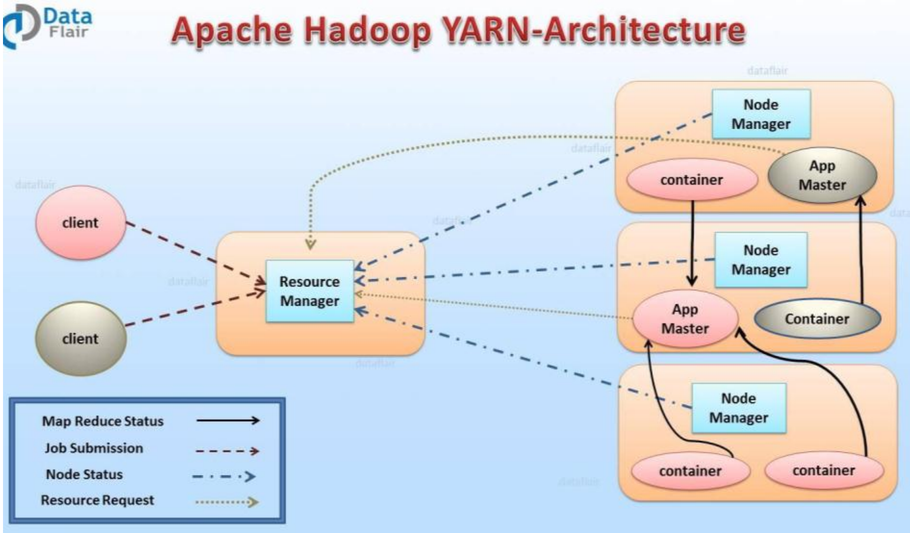


Figure 2.5 Apache Hadoop Yarn Architecture

In this section of Hadoop Yarn, we will discuss the complete architecture of Yarn. Apache Yarn Framework consists of a master daemon known as “Resource Manager”, slave daemon called node manager (one per slave node) and Application Master (one per application).

- **Resource Manager (RM):**

It is Yarn's master daemon. The global allocation of resources (CPU and memory) across all apps is managed by RM. It resolves conflicts over system resources between competing programs. To study Yarn Resource Manager in depth, see to the Resource Manager guide. The main components of Resource Manager are:

✔ Application manager

✔ Scheduler

Application Manager:

It handles the running of Application Masters in the cluster, which means it is in charge of initiating application masters as well as monitoring and restarting them on various nodes in the event of a failure.

Scheduler:

The scheduler is in charge of distributing resources to the currently executing programme. The scheduler is a pure scheduler, which means it conducts no monitoring or tracking for the application and makes no assurances regarding resuming unsuccessful jobs due to application or hardware issues.

- **Node Manager (NM):**

It is Yarn's slave daemon. NM is in charge of monitoring container resource utilization and reporting it to the Resource Manager. On that computer, manage the user process. Yarn Node Manager additionally monitors the node on which it is executing. Long-running auxiliary services can also be plugged into the NM; these are application-specific services that are defined as part of the settings and loaded by the NM upon start-up. A shuffle is a common auxiliary service provided by NMs for MapReduce applications running on YARN.

- **Application Master (AM):**

Each application has its own application master. It interacts with the node manager to negotiate resources from the resource management. It is in charge of the application life cycle.

The AM obtains containers from the RM's Scheduler before contacting the associated NMs to begin the different tasks of the application.

2.6 FEATURES OF YARN

The following characteristics contributed to the popularity of ARN:

- **Scalability:** The scheduler in the YARN architecture's Resource management enables Hadoop to extend and manage thousands of nodes and clusters.
- **Compatibility:** Because YARN supports existing map-reduce applications without interruption, it is also compatible with Hadoop 1.0.
- **Cluster Usage:** Because YARN offers dynamic cluster utilization in Hadoop, optimal Cluster Utilization is possible.
- **Multi-tenancy:** It provides businesses with the benefit of multi-tenancy by allowing multiple engine access.

2.7 APPLICATION WORKFLOW IN HADOOP YARN

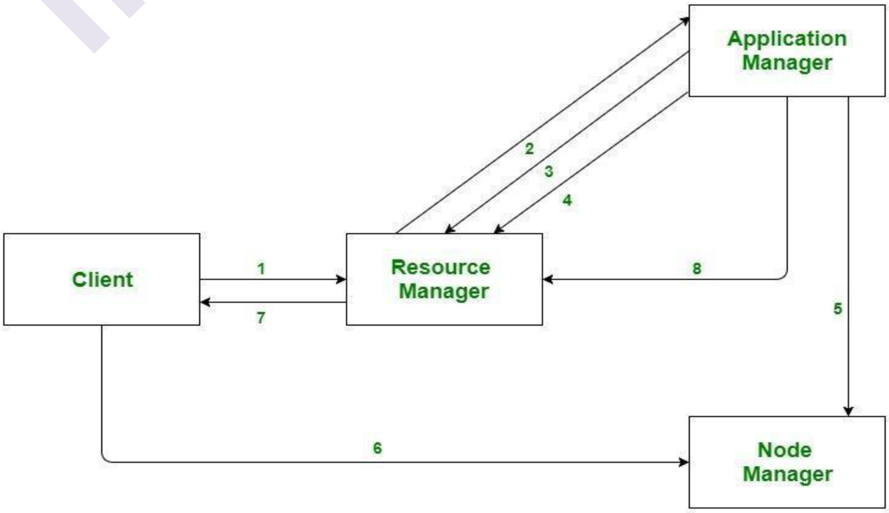


Figure 2.6: Application Workflow

- The client fills out an application.
- The Resource Manager assigns a container to the Application Manager in order for it to run.
- The Resource Manager registers the Application Manager.
- The Application Manager bargains with the Resource Manager for containers.
- The Application Manager alerts the Node Manager that containers should be launched.
- The container executes the application code.
- The client calls the Resource Manager/Application Manager to inquire about the progress of the application.
- When the processing is finished, the Application Manager deregisters from the Resource Manager.

2.8 HADOOP ECOSYSTEM

Apache Hadoop is an open-source framework designed to make interaction with large data easier. However, for people unfamiliar with this technology, the issue of what is big data emerges. Big data is a phrase used to describe data sets that cannot be handled efficiently using standard methodologies such as RDBMS. Hadoop has found a home in sectors and businesses that need to work with enormous data volumes that are sensitive and require fast management. Hadoop is a system that allows for the processing of enormous data volumes in the form of clusters. Hadoop, as a framework, is composed of various modules that are backed by a huge ecosystem of technologies.

Hadoop Ecosystem is a platform or a suite that offers a variety of services to handle big data challenges. It covers Apache projects as well as a variety of commercial tools and solutions. Hadoop consists of four essential components: HDFS, MapReduce, YARN, and Hadoop Common. The majority of the tools or solutions are utilised to augment or assist these key components. All of these instruments operate together to deliver services like as data absorption, analysis, storage, and maintenance.

The following are the components that make up a Hadoop ecosystem:

- **HDFS:** Hadoop Distributed File System
- **YARN:** Yet Another Resource Negotiator
- **MapReduce:** Programming based Data Processing
- **Spark:** In-Memory data processing

PIG, HIVE Query-based processing of data services:

- **HBase:** NoSQL Database
- **Mahout, Spark MLLib:** Machine Learning algorithm libraries
- **Solar, Lucene:** Searching and Indexing
- **Zookeeper:** Managing cluster
- **Oozie:** Job Scheduling

Apart from the above-mentioned components, there are many other components too that are part of the Hadoop ecosystem. All these toolkits or components revolve around one term i.e. Data. That's the beauty of Hadoop that it revolves around data and hence making its synthesis easier.

Other Components: Apart from all of these, there are some other components too that carry out a huge task in order to make Hadoop capable of processing large datasets. They are as follows:

- **Solr, Lucene:** These are the two services that perform the task of searching and indexing with the help of some java libraries, especially Lucene is based on Java which allows spell check mechanism, as well. However, Lucene is driven by Solr.
- **Zookeeper:** There was a huge issue of management of coordination and synchronization among the resources or the components of Hadoop which resulted in inconsistency, often. Zookeeper overcame all the problems by performing synchronization, inter-component-based communication, grouping, and maintenance.
- **Oozie:** Oozie simply performs the task of a scheduler, thus scheduling jobs and binding them together as a single unit. There is two kinds of jobs .i.e Oozie workflow and Oozie coordinator jobs. Oozie workflow is the jobs that need to be executed in a sequentially ordered manner whereas Oozie Coordinator jobs are those that are triggered when some data or external stimulus is given to it.

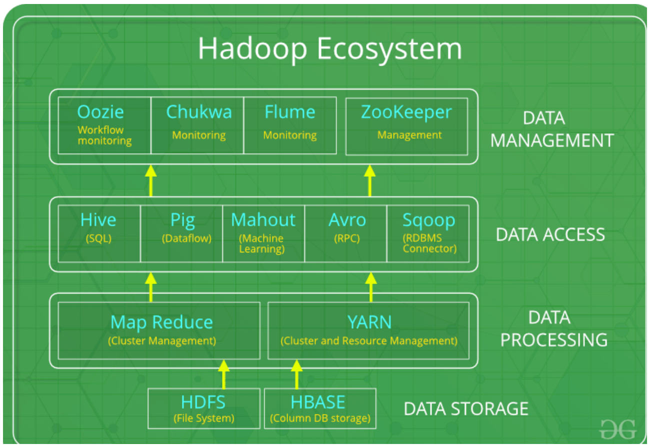


Figure 2.7 Hadoop Ecosystem

2.9 LET US SUM UP

- Hadoop is a framework that uses distributed storage and parallel processing to store and manage big data. It is the software most used by data analysts to handle big data, and its market size continues to grow.
- The three main components of Hadoop are HDFS, Map-reduce, and YARN.
- Data is stored in a distributed manner in HDFS. There are two components of HDFS - name node and data node. While there is only one name node, there can be multiple data nodes.
- Master and slave nodes form the HDFS cluster. The name node is called the master, and the data nodes are called the slaves.

- Master and slave nodes form the HDFS cluster. The name node is called the master, and the data nodes are called the slaves.

2.10 REFERENCES

1. Tom White, "HADOOP: The definitive Guide" O Reilly 2012, Third Edition, ISBN: 978-1-449-31152-0
2. Shiva Achari," Hadoop Essential "PACKT Publications, ISBN 978-1-78439-668-8
3. RadhaShankarmani and M. Vijayalakshmi," Big Data Analytics "Wiley Textbook Series, Second Edition, ISBN 9788126565757
4. Jeffrey Aven," Apache Spark in 24 Hours" Sam's Publication, First Edition, ISBN: 0672338513
5. Bill Chambers and MateiZaharia," Spark: The Definitive Guide: Big Data Processing Made Simple "O'Reilly Media; First edition, ISBN-10: 1491912219;
6. James D. Miller," Big Data Visualization" PACKT Publications'- 10: 1785281941
7. Chuck Lam, "Hadoop in Action", Dreamtech Press 2016, First Edition ISBN:139788177228137

2.11 BIBLIOGRAPHY

1. https://hadoop.apache.org/docs/stable/
2. https://pig.apache.org/
3. https://hive.apache.org/
4. https://spark.apache.org/documentation.html
5. https://help.tableau.com/current/pro/desktop/en-us/default.htm

*****

MODULE II

3

HDFS

Unit Structure

3.0 INTRODUCTION TO HDFS

Data has grown exponentially over the period of time leading world towards big data. To store data which was majorly structured previously was easier as compared to data which is mixture of structured and unstructured also in huge volumes. To store and manage such data we require a network of systems i.e. distributed file systems. As they are network-based, all the complications of network programming add up, thus making distributed file systems more complex than regular disk filesystems. One of the vital challenges is making network fault tolerant that means if any of the node in network fails then it should not cause any data loss.

Hadoop is free. It is Java-based. Hadoop was initially released in 2011. It is part of the Apache project sponsored by the Apache Software Foundation (ASF). It was inspired by Google's MapReduce, a software framework in which an application is broken down into numerous small parts. It was named after its creator's (Doug Cutting) child's stuffed toy elephant Hadoop consists of the Hadoop kernel, MapReduce, the Hadoop Distributed File System (HDFS), and a number of related projects such as Apache Hive, HBase, and Zookeeper. The Hadoop framework is used by major players, including Google, Yahoo, and IBM, largely for applications involving search engines and advertising. The preferred operating systems are Windows and Linux, but Hadoop can also work with BSD and OS X.

Hadoop has introduced a distributed file system called as Hadoop Distributed File Systems which is abbreviated as HDFS.

HDFS provides redundant storage for big data by storing that data across a cluster of cheap, unreliable computers, thus extending the amount of available storage capacity that a single machine alone might have. However, because of the networked nature of a distributed file system, HDFS is more complex than traditional file systems. In order to minimize that complexity, HDFS is based off of the centralized storage architecture.[2]

In principle, HDFS is a software layer on top of a native file system such as ext4 orxfs, and in fact Hadoop generalizes the storage layer and can interact with local file systems and other storage types like Amazon S3. However, HDFS is the flagship distributed file system, and for most programming purposes it will be the primary file system you'll be interacting with. HDFS is designed for storing very large files with streaming data access, and as such, it comes with a few caveats:

- HDFS performs best with a modest number of very large files—for example, millions of large files (100 MB or more) rather than billions of smaller files that might occupy the same volume.
- HDFS implements the WORM pattern. Write once and read a lot. Random writes or file appends are prohibited.
- HDFS is optimized for streaming large file reads rather than random reads or selection.

HDFS is therefore best suited for storing raw input data for computations, intermediate results between computation steps, and final results for the entire operation. do.

Not suitable as a data backend for applications that require real-time updates, interactive data analysis, or record-based transaction support. Instead, HDFS users tend to create large heterogeneous data stores to aid in various calculations and analysis by writing data only once and reading it many times. These repositories are also referred to as "data lakes" because they simply store all data for known issues in a recoverable and fault-tolerant manner.

3.1 HDFS ARCHITECTURE

HDFS is particularly fault-tolerant and is designed to be deployed on low-cost hardware. HDFS gives excessive throughput access to software records and is suitable for programs that have massive datasets. HDFS relaxes some POSIX requirements to permit streaming get right of entry to document system data. HDFS changed into firstly built as infrastructure for the Apache Nutch web search engine undertaking. HDFS is part of the Apache Hadoop core undertaking.

3.1.0 NameNode and DataNodes:

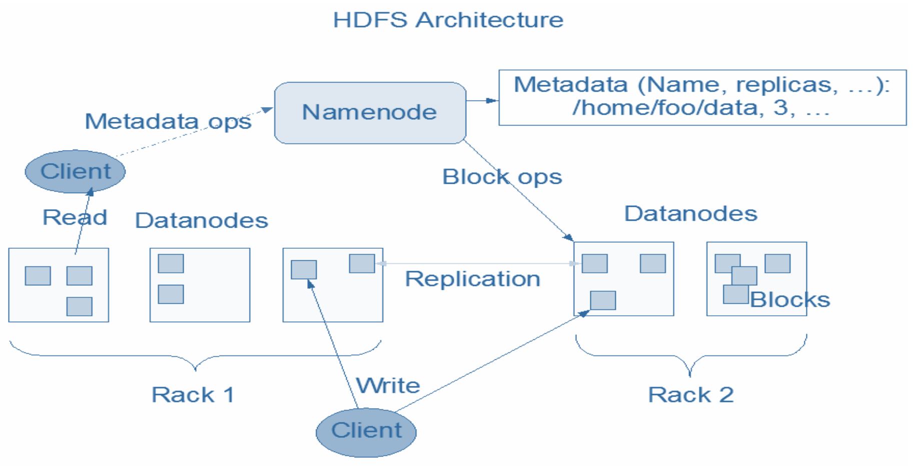


fig: 3.1 HDFS Architecture[1]

HDFS has a master/slave architecture.

An HDFS cluster consists of a single NameNode, a master server that manages the file system namespace and regulates access to files by clients. It maintains the file system tree and metadata for all the files and directories in the tree. This information is stored on the local disk in the form of two files: the namespace image and the edit log.

Along with it there are a some DataNodes, usually one per node in the cluster, which manage storage attached to the nodes that they run on. The Data Nodes are also called as worker nodes. HDFS exposes a file system namespace and allows user data to be stored in files. Internally, a file is split into one or more blocks and these blocks are stored in a set of DataNodes.

The NameNode also knows the DataNodes on which all the blocks for a given file are located, however, it does not store block locations, since this information is reconstructed from DataNodes when the system starts. The NameNode executes file system namespace operations like opening, closing, and renaming files and directories. It also determines the mapping of blocks to DataNodes.

The DataNodes are responsible for serving read and write requests from the file system's clients. The DataNodes also perform block creation, deletion, and replication upon instruction from the NameNode.

It is important to make NameNode fault tolerant as without the NameNode the file system cannot be used. To make is fault tolerant the first step will be taking back-ups for the persistent state files and metadata present in it. Hadoop can be configured so that the NameNode writes its persistent state to multiple filesystems. These writes are synchronous and atomic. The usual configuration choice is to write to local disk as well as a remote NFS mount.

It is also possible to run a secondary NameNode, which has the same name but it still does not act as a NameNode. Its main role is to periodically merge the namespace image with the edit log to prevent the edit log from becoming too large. The secondary NameNode usually runs on a separate physical machine, since it requires plenty of CPU and as much memory as the NameNode to perform the merge. It keeps a copy of the merged namespace image, which can be used in the event of the NameNode failing. However, the state of the secondary NameNode lags that of the primary, so in the event of total failure of the primary, data loss is almost certain. The usual course of action in this case is to copy the NameNode's metadata files that are on NFS to the secondary and run it as the new primary.

3.1.1 The File System Namespace:

HDFS supports a traditional hierarchical file organization. A user or an application can create a directory and store files in that directory. The namespace hierarchy of a file system is similar to most other file systems that exist. You can create and delete files, move files from one directory to another, or rename files. HDFS supports user quotas and permissions. HDFS does not support hard or soft links. However, the HDFS architecture does not preclude the implementation of these features.

HDFS follows the file system naming convention, but some paths and names are reserved, such as /.reserved and .snapshot . Features such as transparent encryption and snapshots use reserved paths.

NameNode maintains the file system namespace. Any changes to the file system namespace or its properties are logged by the NameNode. Applications can specify the number of file replicas to maintain on HDFS. The number of copies of a file is called the replication factor of that file. This information is stored in NameNode.

3.1.2 Data Replication:

The HDFS block size is 64 MB by default, but many clusters use 128 MB (134,217,728 bytes) or even 256 MB (268,435,456 bytes) to ease memory pressure on the namenode and to give mappers more data to work on. Set this using the *dfs.block.size* property in *hdfs-site.xml*.

HDFS is designed to securely store very large files on computers in large clusters. Save each file as a block sequence. Blocks of files are duplicated for fault tolerance. Block size and replication factor are configurable for each file.

All blocks except the last block are the same size, but users can start a new block without filling the last block with the configured block size after variable length block support for append and hsync has been added.

Allows the application to specify the number of copies of a file. Replication factors can be specified during file creation and changed later.

A file in HDFS is written once (except appended and truncated) and has exactly one record at any time.

NameNode makes all decisions regarding block replication. Periodically receive Heartbeat and Blockreport messages from each data node in the cluster. Getting a heartbeat means the DataNode is working properly. Blockreport contains a list of all blocks in the DataNode.

1.1.3 Working with Distributed File System:

Basic File System Operations

To see the available commands in the fs shell, type:

```
hostname $ hadoop fs -help
```

```
Usage: hadoop fs [generic options]
```

As you can see, many of the familiar commands for interacting with the file system are there, specified as arguments to the hadoop fs command as flag arguments in theJava style—that is, as a single dash (–) supplied to the command. Secondary flags or options to the command are specified with additional Java style arguments delimited by spaces following the initial command. Be aware that order can matter when specifying such options. Consider a file named as shakespeare.txt

To copy the above file from one location to another type:

```
hostname $ hadoop fs –copyFromLocal shakespeare.txt
shakespeare.txt
```

In order to better manage the HDFS file system, create a hierarchical tree of directories just as you would on your local file system:

```
hostname $ hadoop fs -mkdir corpora
```

To list the contents of the remote home directory, use the ls command:

```
hostname $ hadoop fs –ls
```

```
drwxr-xr-x - analyst analyst 0 2015-05-04 17:58 corpora

-rw-r--r-- 3 analyst analyst 8877968 2015-05-04 17:52 shakespeare.txt
```

Other basic file operations like mv, cp, and rm will all work as expected on the remote file system. There is, however, no rmdir command; instead, use rm –R to recursively remove a directory with all files in it.

To read the contents of a file, use the cat command, then pipe the output to less in order view the contents of the remote file:

```
hostname $ hadoop fs –cat shakespeare.txt | less
```

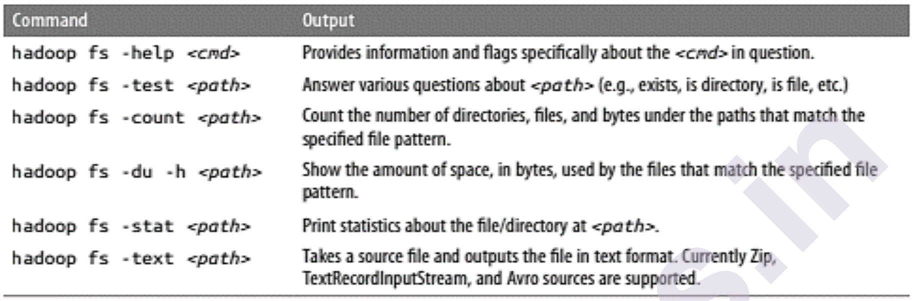
Alternatively, you can use the tail command to inspect only the last kilobyte of the file:

hostname $ hadoop fs –tail shakespeare.txt | less

To transfer entire files from the distributed file system to the local file system, use get or copyToLocal, which are identical commands. Similarly, use the moveToLocal command, which also removes the file from the distributed file system. Finally, the get merge command merges all files that match a given pattern or directory are copied and merged into a single file on the localhost. If files on the remote system are large, you may want to pipe them to a compression utility:

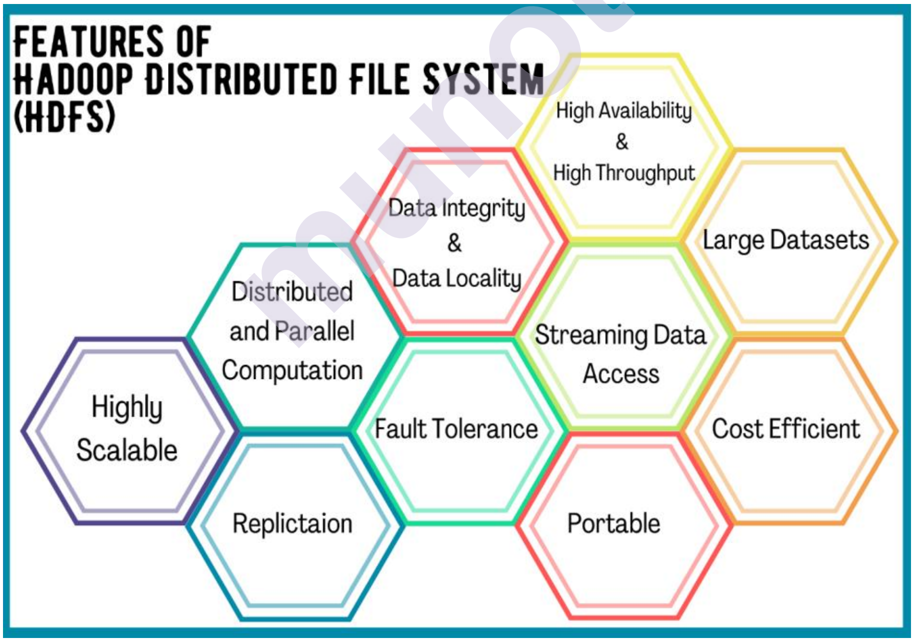
hostname $ hadoop fs –get shakespeare.txt ./shakespeare.from-remote.txt

Other useful utilities



Command	Output
`hadoop fs -help <cmd>`	Provides information and flags specifically about the *<cmd>* in question.
`hadoop fs -test <path>`	Answer various questions about *<path>* (e.g., exists, is directory, is file, etc.)
`hadoop fs -count <path>`	Count the number of directories, files, and bytes under the paths that match the specified file pattern.
`hadoop fs -du -h <path>`	Show the amount of space, in bytes, used by the files that match the specified file pattern.
`hadoop fs -stat <path>`	Print statistics about the file/directory at *<path>*.
`hadoop fs -text <path>`	Takes a source file and outputs the file in text format. Currently Zip, TextRecordInputStream, and Avro sources are supported.

3.2 FEATURES OF HDFS


FEATURES OF
HADOOP DISTRIBUTED FILE SYSTEM
(HDFS)
High Availability & High Throughput
Data Integrity & Data Locality
Large Datasets
Distributed and Parallel Computation
Streaming Data Access
Highly Scalable
Fault Tolerance
Cost Efficient
Replictaion
Portable

The major features of Hadoop Distributed File System (HDFS) are:

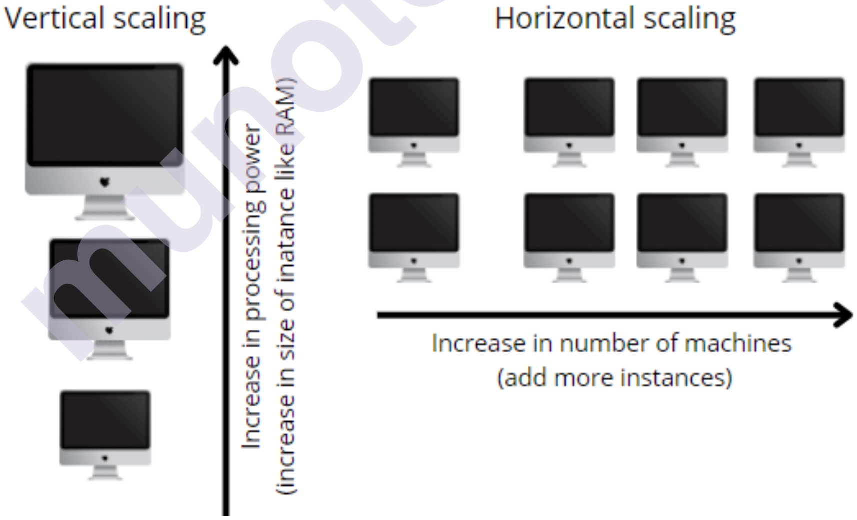
1. Distributed and Parallel Computation:

This is one of the most important features of the Hadoop Distributed File System (HDFS) and makes Hadoop a very powerful tool for big data storage and processing. Here, input data is divided into blocks called data blocks and stored on different nodes in the HDFS cluster.

2. Highly scalable:

There are basically two types of scaling: vertical and horizontal.

- Increases the hardware capacity of the system with scale-up (**vertical scaling)**. That is, add more memory, RAM, and CPU power to an existing system, or buy a new system with more memory, RAM, and CPU. However, there are many problems with this vertical scaling or scaling. First, there is always a limit to what you can increase hardware performance. So you simply cannot keep increasing the amount of memory, RAM or CPU on your machine. Second, when scaling, you must first stop the machine and then add resources to the existing machine. Reboot the system when you increase the performance of the device. Downtime that brings the system down is an issue.
- When you scale out (**horizontal scaling**), you add more nodes to an existing HDFS cluster instead of increasing the system's hardware capacity. And most importantly, you can add more machines on the go. without stopping the system. So, with horizontal scaling there is no downtime. Here you can run more machines in parallel to achieve the main Hadoop goal.


Vertical scaling
Horizontal scaling
Increase in processing power
(increase in size of inatance like RAM)
Increase in number of machines
(add more instances)

3. Replication:

Data Replication is one of the most important and unique features of HDFS. In HDFS replication of data is done to solve the problem of data loss in unfavorable conditions like crashing of a node, hardware failure, and so on.

The data is replicated across a number of machines in the cluster by creating replicas of blocks. The process of replication is maintained at regular intervals of time by HDFS and HDFS keeps creating replicas of user data on different machines present in the cluster. Hence whenever any machine in the cluster gets crashed, the user can access their data from other machines that contain the blocks of that data. Hence there is no possibility of a loss of user data.

4. Fault tolerance:

HDFS is highly fault tolerant and reliable. HDFS creates replicas of file blocks depending on the replication factor and stores them on different machines.

If one of the machines containing data blocks fails, another data node containing copies of these data blocks becomes available. This guarantees data loss and makes the system stable even under adverse conditions.

Hadoop 3 introduces erasure coding for fault tolerance. Erasure Coding in HDFS improves storage efficiency by providing the same level of resiliency and data reliability as traditional replication-based HDFS deployments.

5. Streaming Data Access:

The write-once/read-many design is intended to facilitate streaming reads.

Hadoop Streaming acts as an interface between Hadoop and the program written

6. Portable:

HDFS is designed in such a way that it can easily portable from platform to another.

7. Cost effective:

In HDFS architecture, the DataNodes, which stores the actual data are inexpensive commodity hardware, thus reduces storage costs.

8. Large Datasets:

HDFS can store data of any size (ranging from megabytes to petabytes) and of any formats (structured, unstructured).

9. High Availability:

Hadoop's high availability feature ensures data availability even if a NameNode or DataNode fails. HDFS creates copies of data blocks, so if one of the data nodes goes down, users can access the data on another data node that contains a copy of the same data block.

Also, if the active NameNode fails, the passive node takes over the responsibility of the active NameNode. In this way, data can be accessed and used by users during machine failure.

10. High Throughput:

HDFS stores data in a distributed fashion, allowing parallel processing of data across clusters of nodes. This reduces processing time and ensures high throughput.

11. Data Integrity:

Data integrity refers to the correctness of data. HDFS ensures data integrity by continuously checking data against checksums computed as files are written.

When reading a file, if the checksum does not match the original checksum, the data is said to be corrupted. The client then chooses to receive the data block from another DataNode that has a copy of that block. NameNode discards bad blocks and creates additional new copies.

12. Data Locality:

Data locality means moving computation logic to the data rather than moving data to the computational unit. Data Locality means it co-locate the data with the computing nodes.

In the traditional system, the data is brought at the application layer and then gets processed. But in the present scenario, due to the massive volume of data, bringing data to the application layer degrades the network performance.

In HDFS, we bring the computation part to the Data Nodes where data resides. Hence, with Hadoop HDFS, we are not moving computation logic to the data, rather than moving data to the computation logic. This feature reduces the bandwidth utilization in a system.

3.3 RACK AWARENESS

What is a rack?

The Rack is the collection of around 40-50 DataNodes connected using the same network switch. If the network goes down, the whole rack will be unavailable. A large Hadoop cluster is deployed in multiple racks.

What is Rack Awareness in Hadoop HDFS?

Hadoop components are not supported. For example, HDFS block placement provides fault tolerance by using rack awareness to place replicas of one block in another rack. This ensures data availability in the event of a network switch or partition failure within the cluster.

Hadoop master daemon obtains the cluster worker rack ID by calling an external script or Java class as specified in the configuration file. If you use Java classes or external scripts for your topology, the output must conform to the Java interface **org.apache.hadoop.net.DNSToSwitchMapping**. The interface expects a one-to-one correspondence to be maintained and the topology information in the format of '/myrack/myhost', where '/' is the topology delimiter, 'myrack' is the rack identifier, and 'myhost' is the individual host. Assuming a single /24 subnet per rack, one could use the format of '/193.168.100.0/193.168.100.5' as a unique rack-host topology mapping.

To use the java class for topology mapping, the class name is specified by the **net.topology.node.switch.mapping.impl** parameter in the configuration file. For example, NetworkTopology.java is included in the Hadoop distribution and can be customized by the Hadoop administrator. Using Java classes instead of external scripts has performance benefits because Hadoop does not have to fork external processes when new worker nodes are registered.

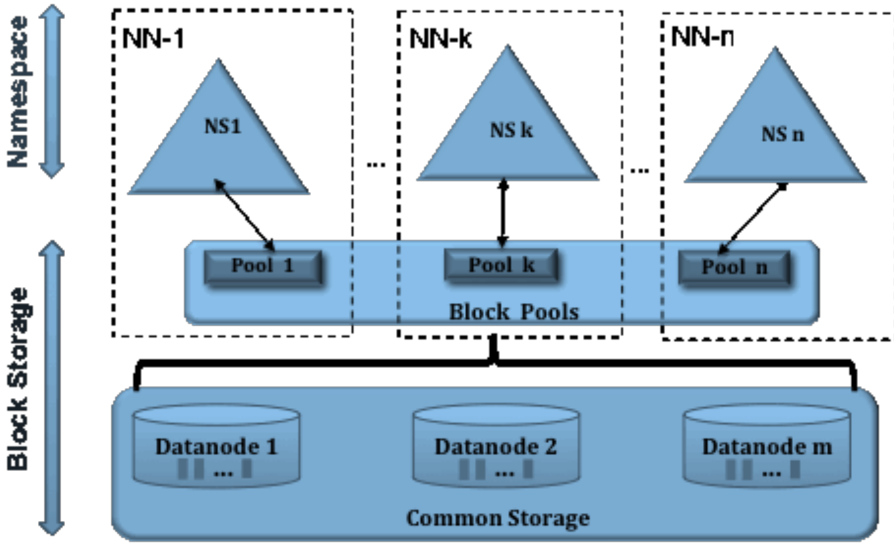
When implementing an external script, it is specified as the **net.topology.script.file.name** parameter in the configuration file. Unlike Java classes, external topology scripts are not included in the Hadoop distribution and are provided by the administrator. Hadoop sends multiple IPs to ARGV when forking topology scripts. The number of IP addresses sent to the topology script is controlled by the **net.topology.script.number.args** setting, which defaults to 100. For **net.topology.script.number.args** is changed to 1, the topology script will branch for each IP address represented by the data node and/or node manager.

If **net.topology.script.file.name** or **net.topology.node.switch.mapping.impl** is not set, the rack ID '/defaultrack' is returned for all forwarded IP addresses. Although this behaviour seems desirable, it can cause problems with HDFS block replication because the default behaviour is to write one replicated block outside the rack, which cannot be done because there is only one rack named "/defaultrack".

Why Rack Awareness?

The reasons for the Rack Awareness in Hadoop are:

1. To reduce the network traffic while file read/write, which improves the cluster performance.
2. To achieve fault tolerance, even when the rack goes down (discussed later in this article).
3. Achieve high availability of data so that data is available even in unfavourable conditions.
4. To reduce the latency, that is, to make the file read/write operations done with lower delay.



The NameNode keeps a reference to every file and block in the filesystem in memory, which means that on very large clusters with many files, memory becomes the limiting factor for scaling.

HDFS Federation, introduced in the 0.23 release series, allows a cluster to scale by adding NameNodes, each of which manages a portion of the filesystem namespace. For example, one NameNode might manage all the files rooted under */user*, say, and a second NameNode might handle files under */share*.

Under federation, each NameNode manages a *namespace volume*, which is made up of the metadata for the namespace, and a *block pool* containing all the blocks for the files in the namespace. Namespace volumes are independent of each other, which means NameNodes do not communicate with one another, and furthermore the failure of one NameNode does not affect the availability of the namespaces managed by other NameNodes. However, block pool storage is not partitioned, so data nodes register with all NameNodes in the cluster and store blocks from multiple block pools. To access a HDFS federated cluster, the client uses a client-side mount table to map file path to a name node. This is controlled in the configuration by the *ViewFileSystem* and *viewfs://* URIs.

3.5 LIST OF REFERENCES

- [1]Apache Hadoop 3.3.2 – HDFS Architecture
- [2] This was first described in the 2003 paper by Ghemawat, Gobioff, and Leung, “The Google File System”.
- Apache Hadoop 3.3.2 – Rack Awareness
- Tom White, “HADOOP: The definitive Guide” O Reilly 2012, Third Edition, ISBN: 978-1-449-31152-0
- What is HDFS? Design Features of HDFS - VTUPulse

- Rack Awareness in Hadoop HDFS - An Introductory Guide - DataFlair (data-flair.training)

3.6 QUIZ

1. HDFS works in a __________ fashion.

   a) worker-master fashion

   b) master-slave fashion

   c) master-worker fashion

   d) slave-master fashion

2. ________ NameNode is used when the Primary NameNode goes down.

   a) Secondary

   b) Rack

   c) Data

   d) Proxy

3. The minimum amount of data that HDFS can read or write is called a _________.

   a) Data Node

   b) Name Node

   c) Rack

   d) Block

4. The default block size is ______.

   a) 32 MB

   b) 16 MB

   c) 128MB

   d) **64 MB**

5. Which of the following is not Features Of HDFS?

   a) **Hadoop does not provide scalability**

   b) It is suitable for the distributed storage and processing.

   c) Streaming access to file system data.

   d) HDFS provides file permissions and authentication

6. Data in ___________ bytes size is called Big Data.

   a) Tera

   b) **Peta**

   c) Giga

   d) Mega

7. Which of the following are false about Hadoop?

   a) Hadoop works in Master-Slave fashion

   b) Master & Slave both are worker nodes

   c) Slaves are actual worker nodes

   d) User submits his work on master, which distribute it to slaves

8. Under HDFS federation

   a) **Each namenode manages metadata of a portion of the filesystem.**

   b) Each namenode manages metadata of the entire filesystem.

   c) Failure of one namenode causes loss of some metadata availability from the entire filesystem.

   d) Each datanode registers with each namenode.

9. A _____ contains all the blocks for the files in the namespace.

   a) block pond

   b) **block pool**

   c) block manager

   d) block group

10. _____________ acts as an interface between Hadoop and the program written?

    a) Hadoop Mapping

    b) Hadoop Clusters

    c) Hadoop Sequencing

    d) Hadoop Streaming

11. What is D in HDFS?

    a) Distributed

    b) Database

    c) Datawarehose

    d) Data

12. Which of the following are correct?

S1: Namespace volumes are independent of each other

S2: Namespace volumes are managed by namenode

    a) S1 only

    b) Both S1 and S2

    c) S2 only

    d) Neither S1 nor S2

13. What is the minimum amount of data that a disk can read or write in HDFS?

    a) block size

    b) byte size

    c) array size

    d) heap size

14. dfs.block.size property in __________

    a) yarn-site.xml

    b) scheduler.xml

    c) hdfs-site.xml

    d) config.xml

15. To list the contents of the remote home directory, use the _______ command

    a) fs

    b) cp

    c) ls

    d) listed

16. Use _______ to recursively remove a directory with all files in it.

    a) rm –R

    b) rmo –Z

    c) rmv –A

    d) rem –K

17. _________ show the amount of space, in bytes, used by the files that match the specified file pattern.

    a) hadoop fs -du -h <path>

    b) hadoop fs -stat -h <path>

    c) hadoop fs -count -h <path>

    d) hadoop fs -test -h <path>

18. To see the available commands in the fs shell

    a) -help

    b) -stat

    c) -all

    d) -show

19. The number of IP addresses sent to the topology script is controlled by the **net.topology.script.number.args** setting, which defaults to ___

    a) **100**

    b) 1000

    c) 10

    d) 1

20. Data locality feature in Hadoop means

    a) Store the same data across multiple nodes.

    b) Relocate the data from one node to another.

    c) Co-locate the data with the computing nodes.

    d) Distribute the data across multiple nodes.

3.7 EXERCISE

1. What is HDFS?
2. Explain HDFS architecture with a neat labelled diagram.
3. What are the features of HDFS?
4. Explain Data Replication in detail.
5. Short note on file system namespace.
6. What are the basic HDFS commands?
7. Explain vertical and horizontal scaling.
8. Describe Rack Awareness.
9. Write in brief about HDFS Federation

3.8 VIDEO LINKS

1. (2187) What is HDFS | Hadoop Distributed File System (HDFS) Introduction | Hadoop Training | Edureka - YouTube
2. (2187) Hadoop Distributed File System (HDFS) - YouTube
3. (2187) Design Features of Hadoop Distributed File System (HDFS)- Big Data Analytics 17CS82 by Mahesh Huddar - YouTube
4. (2187) Block Replication in Hadoop Distributed File System (HDFS) - Big data Analytics by Mahesh Huddar - YouTube
5. (2187) Tutorial 7- What Is Scaling In Scaling Out- Big Data Tutorial - YouTube
6. HDFS Federation Tutorial | Hadoop Architecture Tutorial | Hadoop 2.x Cluster Architecture | Edureka - YouTube
7. HADOOP Tutorial for Beginners - HDFS Replication and Rack awareness - PART 4 - YouTube
8. (2187) Safe Mode Rack Awareness High NameNode Availability NameNodeFederation CheckPoints Backup Snapshots - YouTube
9. (2187) [Hindi] Rack Awareness in Hadoop, File Blocks Replication Explained - YouTube

*****

4

MAP REDUCE & ALGORITHMS

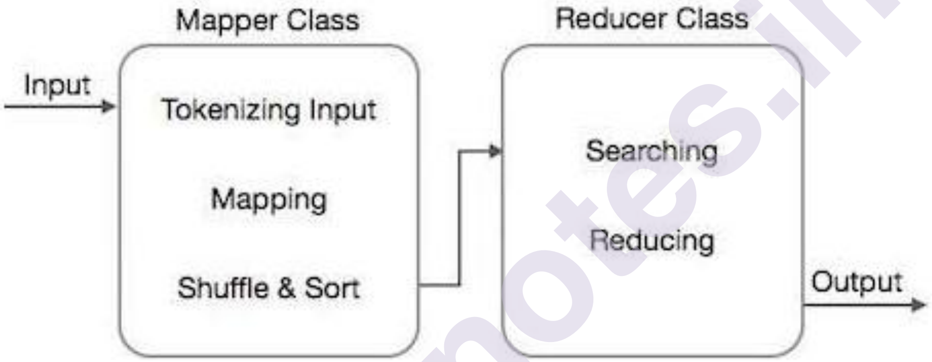
Unit Structure

4.0 INTRODUCTION TO MAP REDUCE

Businesses and governments need to analyse and process vast amounts of data in a very short amount of time. The processing has to be done on large amounts of data and is very time consuming if done on a single machine. So, the idea is to split the data into smaller chunks, send it to a cluster of machines that can be processed concurrently, and then merge the results. The huge growth of data generated by social networks and other blogging sites such as "Friends" on the social network site has increased the amount of graphic data with millions of nodes and edges. This created a new software stack. This new software stack brings parallelism to the using multiple publicly available hardware connected via Ethernet or a switch. Hadoop is a platform for large-scale distributed batch processing. Hadoop can be deployed on a single machine if it can process data, but its main purpose is to efficiently distribute large amounts of data across a set of machines for processing. Hadoop includes a Distributed File System (DFS) that splits the input data and sends pieces of the input data to multiple machines in a particular cluster for storage. The focus of this new software stack is on MapReduce, an advanced programming system. This helps to compute the problem in parallel using all connected machines, so that the output result is obtained in an efficient manner. DFS also provides up to 3x data replication to prevent data loss in case of media failure.

MapReduce is an easy-to-understand paradigm, but not easy to implement, especially in distributed systems. Scheduled according to the workload system.

- Tasks must be monitored and managed to ensure that all errors that occur are handled properly and that tasks continue to run in the event of a partial system failure.
- Inputs must be distributed across the cluster.
- Staged processing of input data should be done on a distributed system, preferably on the same system where the data resides.
- Reduction steps must be performed by collecting the intermediate results of multiple map steps and sending them to the appropriate machine.
- The final output must be available to other users, other applications, or other MapReduce jobs.



Hadoop MapReduce Applications:

Following are some practical applications of MapReduce programs.

E-commerce:

E-commerce companies such as Walmart, Ebay, and Amazon use MapReduce to analyze shopping behavior. MapReduce provides valuable information that is used as a basis for developing product recommendations. Some of the information used includes site history, e-commerce directories, purchase history, and interaction logs.

Social Networks:

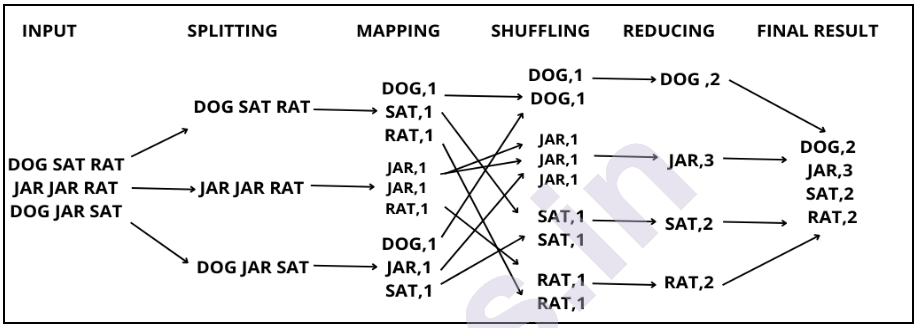
The MapReduce programming tool can evaluate certain information on social media platforms such as Facebook, Twitter and LinkedIn. You can rate important information such as who liked your status and who viewed your profile.

Entertainment:

Netflix uses MapReduce to analyze click and online customer logs. This information helps the company to offer movies based on the interests and behaviors of its customers.

4.1 MAP REDUCE ARCHITECTURE

The MapReduce framework improves job scheduling and monitoring. Failed tasks are re-executed by the framework. This framework is easy to use even for programmers with little experience in distributed processing. MapReduce can be implemented using a variety of programming languages, including Java, Hive, Pig, Scala, and Python.



Map Reduce Architecture

The applications that use MapReduce have the below advantages:

- They have been provided with convergence and good generalization performance.
- Data can be handled by making use of data-intensive applications.
- It provides high scalability.
- Counting any occurrences of every word is easy and has a massive document collection.
- A generic tool can be used to search tool in many data analysis.
- It offers load balancing time in large clusters.
- It also helps in the process of extracting contexts of user location, situations, etc.
- It can access large samples of respondents quickly.

4.1.1 The Map Task:

Split the input data set into chunks in a map operation. The Map Task processes these fragments in parallel. Map to use as input to the reduction problem.

4.1.2 The Reduce Task:

Reducers process the intermediate data from the maps into smaller tuples, that reduces the tasks, leading to the final output of the framework.

A developer can always set the number of the reducers to zero. That will completely disable the reduce step.

4.1.3 Grouping by Key:

In the MapReduce process, the mapper only understands the key-value pairs in the data, so before passing the data to the mapper, you must first transform the data into key-value pairs. In Hadoop MapReduce,key-value pairs are generated as follows using InputSplit and Record Reader.

By default, RecordReader uses TextInputFormat to convert data into key/value pairs. The RecordReader interacts with the InputSplit until it has finished reading the file.

In MapReduce, the map function processes a specific key-value pair and produces a specific number of key-value pairs, whereas the Reduce function processes the values grouped by the same key and produces a different set of key-value pairs as output. The output type of the map must match the input type of reduce as shown below.

Map: (K1, V1) > List(K2, V2)

Decrease: {(K2, List(V2 }) > List K3, V3)

Hadoop's generated key-value pairs depend on the dataset and desired output It varies, and typically key-value pairs are specified in four places: Map input, Map output, Reduce input, and Reduce output.

- **Map Input:**

Map-input by default will take the line offset as the key and the content of the line will be the value as Text.

- **Map Output:**

Map basic responsibility is to filter the data and provide the environment for grouping of data based on the key.

- o **Key:** It will be the field/ text/ object on which the data has to be grouped and aggregated on the reducer side.
- o **Value:** It will be the field/ text/ object which is to be handled by each individual reduce method.

- **Reduce Input:**

The output of Map is the input for reduce, so it is same as Map-Output.

- **Reduce Output:**

It depends on the required output.

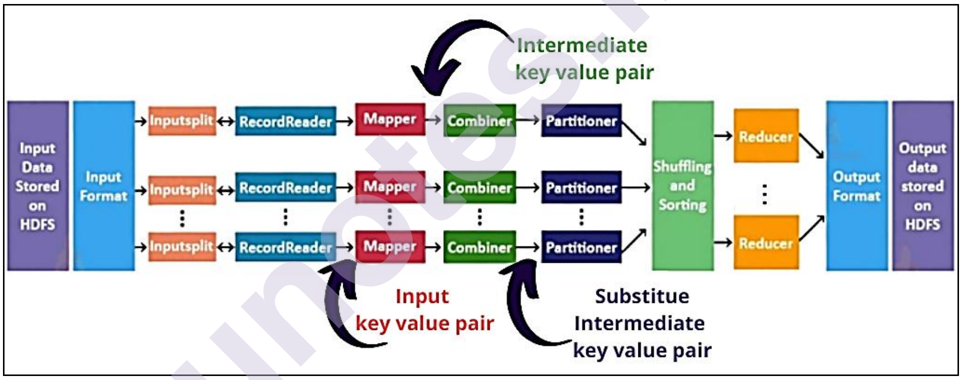
4.1.4 Partitioner and Combiners:

A partitioner partitions the key-value pairs in the Map intermediate output. Separate data using user-defined conditions that act like hash functions. The total number of sections equals the number of Reducer jobs per job.

A Combiner is also called and semi-reducer, is an optional step used to optimize the MapReduce process. Used to reduce output at node level. At this point, you can merge the cloned output from the map output into a single output file. The merge step speeds up the shuffle step by improving job performance.

Combiner class is used between Map class and Reduce class to reduce the amount of data transfer between Map and Reduce. In general, the output of the map job is large and the data passed to the reduce job is large.

4.1.5 Detail of Map Reduce Execution:



The data flow in MapReduce is the combination of different processing phases of such as Input Files, InputFormat in Hadoop, InputSplits, RecordReader, Mapper, Combiner, Partitioner, Shuffling and Sorting, Reducer, RecordWriter, and OutputFormat. Hence all these components play an important role in the Hadoop mapreduce working.

MapReduce processes the data in various phases with the help of different components. The steps of job execution in Hadoop.

Input Files:

The data for MapReduce job is stored in Input Files. Input files reside in HDFS. The input file format is random. Linebased log files and binary format can also be used.

InputFormat:

After that InputFormat defines how to divide and read these input files. It selects the files or other objects for input. InputFormat creates InputSplit.

InputFormat class calls the getSplits function and computes splits for each file and then sends them to the jobtracker.

InputSplits:

It represents the data that will be processed by an individual Mapper. For each split, one map task is created. Therefore, the number of map tasks is equal to the number of InputSplits. Framework divide split into records, which mapper process.

RecordReader:

It communicates with the InputSplit. And then transforms the data into key-value pairs suitable for reading by the Mapper. The RecordReader uses TextInputFormat by default to convert data into key/value pairs. Interact with InputSplit until the file is finished reading. Allocates an offset in bytes for each line in the file. These key/value pairs are then sent to the resolver for further processing.

Mapper:

It processes the input records generated by the RecordReader and generates intermediate key-value pairs. The intermediate output is completely different from the input pair. The result of the transformer is a complete group of key-value pairs. The Hadoop infrastructure does not store the output of the converter in HDFS. The mapper doesn't store it because the data is temporary and creates multiple non-write-free copies on HDFS. The mapper then passes the output to the combiner for further processing.

Combiner:

The Combiner is a Minireducer that performs local aggregation of mapper outputs. This minimizes data transfer between mappers and reducers. So when the combiner function completes, the platform passes the output to the splitter for further processing.

Partitioner:

Partitioner occurs when working with more than one reducer. Capture the output of the combiner and perform the split.

Output splits based on keys in MapReduce. Using a hash function, a key (or a subset of a key) creates partitions.

MapReduce splits each output of the combiner according to the key value. Then records with similar key values belong to the same section. After that, each section is sent to the reducer.

MapReduce splitting ensures that the map output is evenly distributed across the reducer.

Shuffle and Sort:

After splitting, the output is shuffled into collapsed nodes. Shuffling is the physical transfer of data over a network. This is because all converters terminate and shuffle their output at the reducer node. The framework then joins and sorts these intermediate results. This serves as an input to the phase reduction.

Reducer:

The reducer then takes as input the intermediate set of key-value pairs generated by the converter. It then runs the reducer function on each to produce an output. The gear output is the decisive output. The platform then saves the output to HDFS.

How many Reduces?

The right number of reduces seems to be 0.95 or 1.75 multiplied by (<no. of nodes> * <no. of maximum containers per node>).

With 0.95 all of the reduces can launch immediately and start transferring map outputs as the maps finish. With 1.75 the faster nodes will finish their first round of reduces and launch a second wave of reduces doing a much better job of load balancing.

Increasing the number of reduces increases the framework overhead, but increases load balancing and lowers the cost of failures.

The scaling factors above are slightly less than whole numbers to reserve a few reduce slots in the framework for speculative-tasks and failed tasks.

RecordWriter:

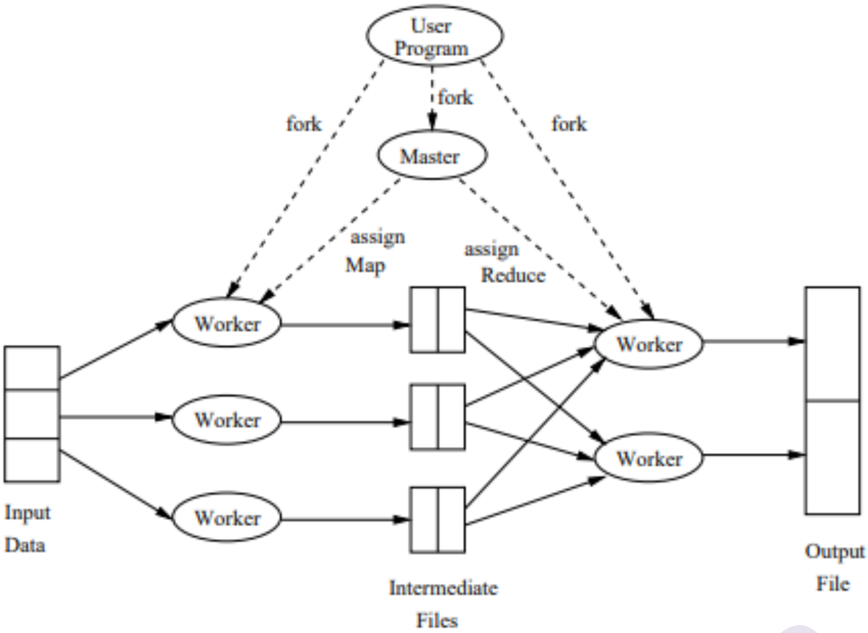
Writes this output key-value pair to the output file from the reducer stage.

OutputFormat:

OutputFormat specifies how the RecordReader writes these output key/value pairs to the output file. So, the instance served by Hadoop writes files to HDFS. So, the OutputFormat instance writes the reducer's deterministic output to HDFS.

Typically, a worker processes either a Map job (Map worker) or a Reduce job (Reduce worker), but not both jobs at the same time.

A master has many responsibilities. One is to create multiple Map jobs and Reduce jobs that the user program selects. These tasks are assigned to the workflow by the master. It makes sense to create one Map job for each chunk of the input file, but you may want to create fewer Reduce jobs. The reason for limiting the number of reduce jobs is that intermediate files must be created for each map job, and if there are too many reduce jobs, the number of intermediate files increases.



The Master keeps track of the state of each Map and Reduce job (idle, running in a specific worker process, or finished). The workflow reports to the master when the task is complete, and the master assigns a new task to that workflow.

Each map job is assigned one or more chunks of an input file and runs user-written code. The Map task creates a file for each Reduce task on the local disk of the worker running the Map task. The master receives information about the location and size of each of these files and the shrink operation of those files. When the master assigns a collapse action to a job flow, all the files that make up the input are passed to that job. The Reduce job runs user-written code and writes the results to a file that is part of the surrounding distributed file system.

4.2 ALGORITHM USING MAP REDUCE

MapReduce is a distributed data processing algorithm introduced by Google. The MapReduce algorithm is primarily based on a functional programming model. The MapReduce algorithm is useful for reliably and efficiently processing large amounts of data in parallel in a cluster environment.

Divides input tasks into smaller, more manageable subtasks and executes them in parallel. The MapReduce algorithm is based on sending processing nodes (local machines) to where the data resides.

The MapReduce algorithm works by splitting the process into three steps.

- Matching Stage
- Sorting and Moving Stage
- Reducing Stage

4.2.1 Matrix and Vector Multiplication by Map Reduce:

The original purpose for which the Google MapReduce implementation was created was to perform the very large matrix-vector multiplication required to compute the PageRank. It can be seen that matrix-vector and matrix-matrix computations are well suited for MapReduce-style computing. Another important class of operations where MapReduce can be effectively used is relational algebra operations.

Suppose we have an n × n matrix M, in which the entries in row i and column j are represented by m_{ij}. Suppose we have a vector v of length n whose jth element is equal to v_j. Then the matrix-vector product is a vector x of length n where the i-th element xi is

$$x_i = \sum_{j=1}^{n} m_{ij} u_j$$

If n = 100, we don't want to use DFS or MapReduce for this calculation.

However, this kind of calculation is the basis of the web page rankings performed by search engines, and n is in the tens of millions, so it can be used for any map operation.

Matrix M and vector v are stored in DFS files. Assume that the row column coordinates of each matrix element can be found either by their location in a file, or because they are stored in triples (i, j, m_{ij}) with explicit coordinates. We also assume that the position of the element v_j in the vector v will be found in a similar way.

Map Function: The Map function is written to be applied to a single element M. However, if v has not yet been read into the main memory of the compute node executing the Map operation, then v will first read the whole, then will be available to all applications of the "Map" function performed in this "Map" operation. Each map operation operates on a slice of matrix M. A key-value pair (i, $m_{ij}v_j$) is generated from each element of the m_{ij} matrix. Thus, all members of the sum that make up the xi component of the matrix-vector product receive the same key.

Reduce function: The reduce function simply sums up all values associated with a given key i. The result is a pair (i, x_i).

4.2.2 Computing Selection and Projection by Map Reduce:

Computing Selection by Map Reduce:

Selection does not require all the features of MapReduce. This can be done most conveniently only on the map part, but it can also be done only on the reduced part. Here is a MapReduce implementation of σC(R) allocation.

The Map function: check that C is satisfied for each tuple t in R. If so, create key-value pairs (t, t). That is, both the key and the value are t.

The Reduce Function: Reduce function is an identity. It simply passes of each key-value pair as output.

Computing Projection by Map Reduce:

Projection is performed similarly to selection, because projection may cause the same tuple to appear several times, the Reduce function must eliminate duplicates. We may compute πS (R) as follows.

The Map Function: For each tuple t in R, construct a tuple t′ by eliminating from t those components whose attributes are not in S. Output the key-value pair (t′, t′).

The Reduce Function: For each key t′ produced by any of the Map tasks, there will be one or more key-value pairs (t′, t′). The reduce function converts (t', [t', t',..., t']) to (t',t') so that for that key t' we get exactly one pair (t', t'). create.

Note that shrink operation is deduplication. Because these operations are associative and commutative, the combiner associated with each map operation can remove locally generated duplicates. But you still need a Reduce job to remove two identical tuples coming from different Map jobs.

4.2.3 Computing Grouping and Aggregation by Map Reduce:

As with joins, a minimal grouping and aggregation example with one grouping property and one aggregation is described. Let R(A, B, C) be a relation to which the operators γA, θ(B)(R) are applicable. Map does group and Reduce does aggregation.

The Map function: Creates a key-value pair (a, b) for each tuple (a, b, c).

The Reduce Function: Each key represents a group. Aggregate operator θ list [b1, b2,.,bn] value B associated with key a. Output is the (a, x) pair. where x is the result of applying θ to the list. For example, if θ is equal to SUM, then x = b1 + b2 + · · · + bn, and if θ is equal to MAX, then x is b1, b2, . . . ,billion.

If there are multiple grouping attributes, the key is a list of tuple values for all these attributes. If there is more than one aggregation, the reduction function applies each to the list of values associated with the given key and returns a tuple of keys with the component for all properties. Grouping (each aggregation result if more than one).

4.3 SELF-LEARNING TOPIC: CONCEPT OF SORTING AND NATURAL JOINS

Concept of Sorting:

Sorting is one of the basic MapReduce algorithms to process and analyze data. MapReduce implements sorting algorithm to automatically sort the output key-value pairs from the mapper by their keys.Sorting methods are implemented in the mapper class itself. In the Shuffle and Sort phase, after tokenizing the values in the mapper class, the Context class (userdefined class) collects the matching valued keys as a collection. To collect similar key-value pairs (intermediate keys), the Mapper class takes the help of RawComparator class to sort the key-value pairs. The set of intermediate key-value pairs for a given Reducer is automatically sorted by Hadoop to form key-values (K2, {V2, V2, …}) before they are presented to the Reducer.

Computing Natural Join by MapReduce:

Natural Join: Given two relations, compare each pair of tuples, one from each relation. If the tuples agree on all the attributes that are common to the two schemas, then produce a tuple that has components for each of the attributes in either schema and agrees with the two tuples on each attribute. If the tuples disagree on one or more shared attributes, then produce nothing from this pair of tuples. The natural join of relations R and S is denoted R $\bowtie$ S. While we shall discuss executing only the natural join with MapReduce, all equijoins (joins where the tuple agreement condition involves equality of attributes from the two relations that do not necessarily have the same name) can be executed in the same manner.

The idea of implementing natural joins using MapReduce comes to mind if we consider the specific case of joining R(A, B) with S(B, C). It needs to find a matching tuple in the second component of the R-tuple and the first component of the S-tuple, the B-component, and uses the B-tuple values of all relationships as keys. The values will be the names of other components and relationships, so the reduce function knows where each tuple comes from.

The Map Function: For each tuple (a, b) of R, produce the key-value pair (b,(R,a)). For each tuple (b,c) of S, produce the key-vale pair (b,(S,c)).

The Reduce Function: Each key value b will be associated with a list of pairs that are either of the form (R, a) or (S, c). Construct all pairs consisting of one with first component R and the other with first component S, say (R, a) and (S, c). The output from this key and value list is a sequence of key-value pairs. The key is irrelevant. Each value is one of the triples (a, b, c) such that (R, a) and (S, c) are on the input list of values.

The same algorithm works if the relationship has more than one attribute. You can think of A representing all attributes of schema R, but not S. B represents properties from both schemas, and C represents properties from schema S only. The keys of an R or S tuple are the following list: The value tuple of all attributes in both R and S schemas. The value of R is the name of R, along with the values of all attributes that belong to R but not S. The tuple S is the name of S with attribute values belonging to S but not R. The reduce function goes through all key-value pairs with a given key and combines these values in R with these S values in every possible way.

Of the in each pair, the resulting tuple contains the values in R, the key values, and values in S.

In MapReduce programs, Mapper's output key-value pairs are automatically sorted by key. This feature can be used in programs that require alignment at certain stages. The minimum spanning tree program is an example of using the auto-align feature to sort edges by weight. This example also demonstrates the use of common data structures for reducers. The downside of this data structure is that it doesn't allow MapReduce programs to be fully parallelized. RadhaShankarmani and M. Vijayalakshmi, "Big Data Analytics" Wiley Textbook Series, Second Edition, ISBN 9788126565757

4.5 LIST OF REFERENCES

- Anand Rajaraman and Jerey David Ullman. Mining of Massive Datasets. Cambridge University Press, New York, NY, USA, 2011
- Data Analytics with Hadoop An Introduction for Data Scientists, Benjamin Bengfort and Jenny Kim, O'Reilly, 2016
- Hadoop – Apache Hadoop 3.3.2
- Understanding MapReduce in Hadoop | Engineering Education (EngEd) Program | Section
- How Hadoop MapReduce Works - MapReduce Tutorial - DataFlair (data-flair.training)
- MapReduce Algorithms | A Concise Guide to MapReduce Algorithms (educba.com)
- Learn the Concept of Key-Value Pair in Hadoop MapReduce - DataFlair (data-flair.training)
- Hadoop & Mapreduce Tutorial | Counters, Sorting and Joins (vskills.in)
- Relational Operations Using MapReduce | by Kartikeya Sharma | The Startup | Medium

4.6 QUIZ

1. The MapReduce algorithm contains two important tasks, namely __________.

   a) mapped, reduce

   b) mapping, Reduction

   c) Map, Reduction

   d) Map, Reduce

2. InputFormat class calls the ________ function and computes splits for each file and then sends them to the jobtracker.

   a) getSplits

   b) splits

   c) put

   d) gets

3. _________ function is responsible for consolidating the results produced by each of the Map() functions/tasks.

   a) Map

   b) Reducer

   c) mapper

   d) Reduce

4. Which of the following phases occur simultaneously?

   a) Reduce and Shuffle

   b) Shuffle and Sort

   c) Reduce and Sort

   d) **Shuffle and Sort**

5. The right number of reduces seems to be_

   a) **0.95**

   b) 0.36

   c) 0.90

   d) 0.80

6. Which of the following is true about MapReduce?

   a) It provides the resource management

   b) **Data processing layer of Hadoop**

   c) An open source data warehouse system for querying and analysing large datasets stored in Hadoop files

   d) distributed processing system which utilizes in-memory computing

7. Which of the following are false about Map-Reduce?

   a) Map-Reduce works in Master-Slave fashion

   b) Master & Slave both are worker nodes

   c) Slaves are actual worker nodes

   d) User submits his work on master, which distribute it to slaves

8. Which among the following is the programming model designed for processing large volumes of data in parallel by dividing the work into a set of independent tasks?

   a) MapReduce

   b) HDFS

   c) PIG

   d) YARN

9. Which among the following controls the partitioning of the keys of the intermediate map-outputs?

   a) RecordWriter

   b) **Partitioner**

   c) RecordReader

   d) Combiner

10. What is the correct sequence of data flow in MapReduce?

    a. InputFormat

    b. Mapper

    c. Combiner

    d. Reducer

    e. Partitioner

    f. OutputFormat

    a) abcdfe

    b) acdefb

    c) abcdef

    d) abcedf

11. Which of the following is not the phase of Reducer?

    a) Map

    b) Shuffle

    c) Sort

    d) Reduce

12. Put the following phases of a MapReduce program in the order that they execute?

    a. Partitioner

    b. Mapper

    c. Combiner

    d. Shuffle/Sort

    a) Mapper Partitioner Shuffle/Sort Combiner

    b) Mapper Combiner Partitioner Shuffle/Sort

    c) Mapper Partitioner Shuffle/Sort Combiner

    d) Mapper Shuffle/Sort Combiner Partitioner

13. Which among the following generate an intermediate key-value pair?

    a) Mapper

    b) Reducer

    c) Combiner

    d) Partitioner

14. Which of the following is called Mini-reduce?

    a) Mapper

    b) Reducer

    c) Combiner

    d) Partitioner

15. By default RecordReader uses which InputFormat for converting data into key-value pairs.

    a) FileInputFormat

    b) KeyValueTextInputFormat

    c) TextInputFormat

    d) SequenceFileInputFormat

16. Use _______ to recursively remove a directory with all files in it?

    a) getSplit()

    b) get.InputSplit()

    c) getInputSplit()

    d) getInputSplit(int)

17. Which of the following is the correct sequence of MapReduce flow?

    a) Map>Combine> Reduce

    b) Combine>Reduce>Map

    c) Map>Reduce>Combine

    d) Reduce>Combine>Map

18. To collect similar key-value pairs (intermediate keys), the Mapper class takes the help of _________ class to sort the key-value pairs.

    a) RawComparator

    b) InputComparator

    c) OutputComparator

    d) IntermidiateComparator

19. What is the input to the Reduce function?

    a) **One key and a list of all values associated with that key**.

    b) One key and a list of some values associated with that key.

    c) An arbitrarily sized list of key/value pairs.

    d) One key and a list of all values associated with other key

20. How can you disable the reduce step?

    a) The Hadoop administrator has to set the number of the reducer slot to zero on all slave nodes. This will disable the reduce step.

    b) It is impossible to disable the reduce step since it is critical part of the Mep-Reduce abstraction.

    c) A developer can always set the number of the reducers to zero. That will completely disable the reduce step.

    d) While you cannot completely disable reducers you can set output to one. There needs to be at least one reduce step in Map-Reduce abstraction.

4.7 EXERCISE

1. What is Map Reduce?
2. Explain Map Reduce architecture with a neat labelled diagram.
3. What are the applications and features of Map Reduce?
4. Explain Partitioner and Combiner in detail.
5. Explain Map Reduce execution in detail.
6. Describe Matrix and Vector Multiplication by Map Reduce
7. Explain Computing Selection and Projection by Map Reduce
8. Explain Computing Grouping and Aggregation by Map Reduce
9. Short note on sorting and natural joins

4.8 VIDEO LINKS

1. Map Reduce ll Master Job Tracker and Slave Tracker Explained with Examples in Hindi - YouTube
2. (2220) MapReduce Tutorial | What is MapReduce | Hadoop MapReduce Tutorial | Edureka - YouTube
3. (2220) Mapreduce In Hadoop | MapReduce Explained | MapReduce Architecture | MapReduce Tutorial |Simplilearn - YouTube
4. Map Reduce Paper - Distributed data processing - YouTube
5. What is MapReduce? - YouTube
6. Mapper in MapReduce - YouTube
7. Partitioner in MapReduce - YouTube
8. Combiner in MapReduce - YouTube
9. (2220) Reducer in MapReduce - YouTube
10. (2220) Shuffle & Sorting of MapReduce Task - YouTube
11. (2201) Advanced MapReduce - Join Types | Introduction to MapReduce Joins - YouTube

*****

MODULE III

5

INTRODUCTION TO NOSQL

Unit Structure

5.0 OBJECTIVES

This chapter would make you understand the following concepts:

- What is NoSQL and how NoSQL is an important technology in the big data landscape.
- Features of NoSQL which makes it so popular.
- CAP Theorem and visual guide to NoSQL and CAP theorem.
- NoSQL Business drivers like volume, velocity, agility and variability
- Advantages of NoSQL
- Sharding and Replication along with its advantages
- NoSQL Data Architecture patterns – Key value store, Column Database, Graph Data store, Document-based store
- Use of NoSQL in Industry
- What is Big Data?
- Relation between NoSQL and Big Data
- Understanding various problems which are faced due to Big Data in different industries with the help of multiple use cases.

5.1 WHAT IS NOSQL?

We say that today is the age of Big Data. The sheer volume of data being generated today is exploding. The rate of data creation or generation is mind boggling. Mobile phones, social media, imaging technologies which are used for medical diagnosis, non-traditional IT devices like RFID readers, automated trackers, monitoring systems, GPS navigation systems —all these are among the fastest growing sources of data. Now keeping up with this huge influx of data is difficult, but what is more challenging is storing, analysing and archiving vast amounts of this generated data.

Along with the speedy data growth, data is also becoming increasingly semi-structured and sparse. It means the traditional data management technologies which were designed around upfront schema definition and relational references are no longer useful. This has led to the rise of a newer class of database technologies. The big data technology landscape majorly consists of two important technologies – NoSQL and Hadoop.

The term NoSQL was first used by Carlo Strozzi in 1998 to name his lightweight, open-source relational database that did not expose the standard SQL interface. NoSQL is actually an acronym that stands for 'Not Only SQL'. Today NoSQL is used as an umbrella term for all the data stores and databases that do not follow traditional well established RDBMS principles. NoSQL is a set of concepts that allows the rapid and efficient processing of data sets with primary focus on performance, reliability and agility. NoSQL databases are widely used in big data and other real-time web applications. These are non-relational, open source and distributed databases. They are getting hugely popular because of their ability to scale out or scale horizontally and their proficiency at dealing with a rich variety of structured, semi-structured and unstructured data.

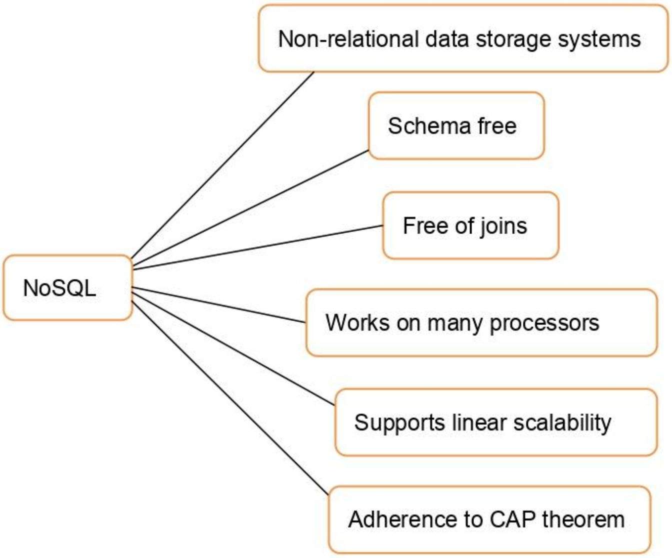
5.2 FEATURES OF NOSQL

Following are the various features of NoSQL which makes it a popular technology:

1. **Non-relational data storage system:** They do not adhere to the relational data model. NoSQL systems store and retrieve data from many formats like key-value pairs or document-oriented or column-oriented or graph-based databases.

2. **Schema free:** NoSQL databases are gaining popularity because of their support for flexibility to the schema. They easily allow data not to strictly adhere to any schema structure at the time of storage.

3. **Free of joins:** NoSQL databases allow users to extract data using simple interfaces without joins.



4. **Works on many processors:** NoSQL systems allow users to store the database on multiple processors and still maintain high-speed performance.

5. **Supports linear scalability:** Scalability is the ability of a system to increase throughput with the addition of resources to address increase in load. There is a consistent increase in the performance after adding multiple processors. NoSQL allows easy scaling to adapt to the data volume and complexity of cloud applications.

6. **Adherence to CAP theorem:** NoSQL databases do not offer support for ACID properties (Atomicity, Consistency, Isolation and Durability). On the contrary, they abide by Brewer's CAP theorem (Consistency, Availability and Partition tolerance) and are often seen compromising on consistency in favour of availability and partition tolerance.

5.3 CAP THEOREM

Eric Brewer first introduced the CAP theorem in year 2000 at a symposium. This theorem is widely adopted today by large web companies like Amazon as well as the NoSQL community. The acronym CAP stands for Consistency, Availability and Partition Tolerance.

- **Consistency:** Making available the same single updated readable version of the data to all the clients. Here, Consistency is concerned with multiple clients reading the same data from replicated partitions and getting consistent results.

- **High Availability:** System remains functional even if some nodes fail. High Availability means that the system is designed and implemented in such a way that it continues its read and write operation even if nodes in a cluster crash or some hardware or software parts are down due to upgrades. Internal communication failures between replicated data should not prevent updates.

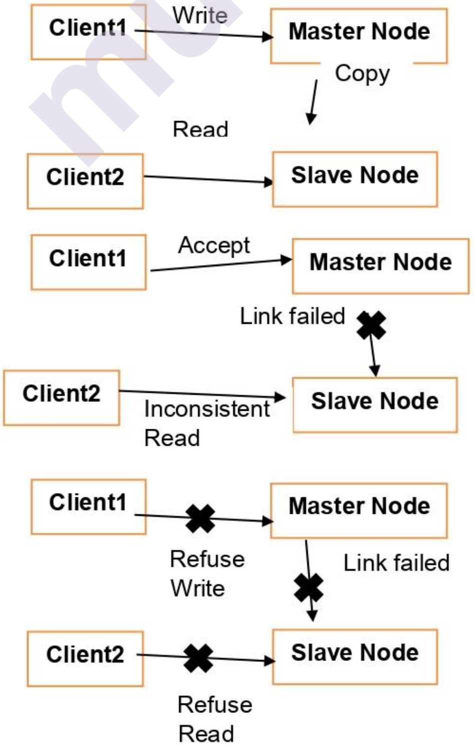
- **Partition tolerance:** Partition tolerance is the ability of the system to continue functioning in the presence of network partitions. This situation arises when network nodes cannot connect to each other (temporarily or permanently). System remains operations on system split or communication malfunction. A single node failure should not cause the entire system to collapse.

Distributed systems can only guarantee two of the features, not all three. Satisfying all three features at the same time is impossible. CAP theorem summarises that if you cannot limit the number of faults and requests can be directed to any server and you insist on serving every request then you cannot possibly remain consistent. It means you must always give up something – either consistency or availability or tolerance to failure and reconfiguration. So, CAP theorem can help the organization's decide what properties (Consistency, Availability or Partition Tolerance) are most important for their business.

If consistency and high availability are simultaneously required then a faster single processor can be the best choice. E.g., Amazon's Dynamo is available and partition tolerant but not strictly consistent whereas Google's Bigtable chooses consistency and availability over partition tolerance.

Choose consistency over availability whenever your business requirements demand atomic read and write operations. Choose availability over consistency when your business requirements allow some flexibility around when the data in the system is in sync.

The below given diagram helps you decide the relative merits of availability versus consistency when a network fails.



Normal Operation

Under normal operation a client write operation will be carried on the master node and then will be replicated over the network onto a slave node.

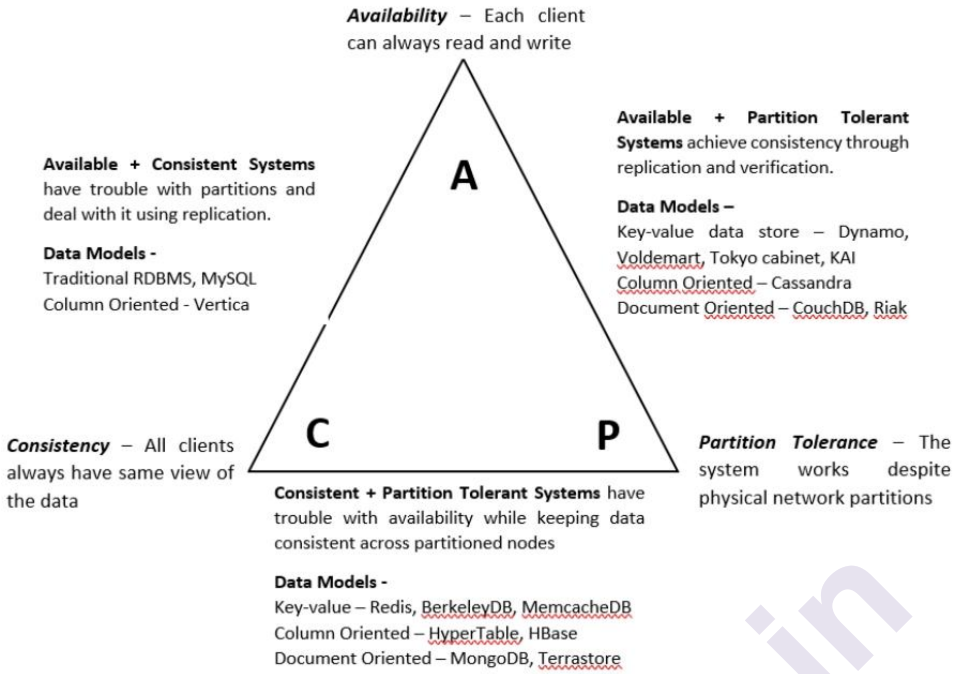
High Availability option

In case of link failure, you accept a write and risk inconsistent reads from the slave.

Consistency option

You choose consistency over availability and so block the client's write operation on the master node until the link between the data centres is restored back.

Following is the visual guide to NoSQL and CAP theorem:



Availability – Each client can always read and write

Available + Consistent Systems have trouble with partitions and deal with it using replication.

Data Models -
Traditional RDBMS, MySQL
Column Oriented - Vertica

Available + Partition Tolerant Systems achieve consistency through replication and verification.

Data Models –
Key-value data store – Dynamo, Voldemart, Tokyo cabinet, KAI
Column Oriented – Cassandra
Document Oriented – CouchDB, Riak

Consistency – All clients always have same view of the data

Partition Tolerance – The system works despite physical network partitions

Consistent + Partition Tolerant Systems have trouble with availability while keeping data consistent across partitioned nodes

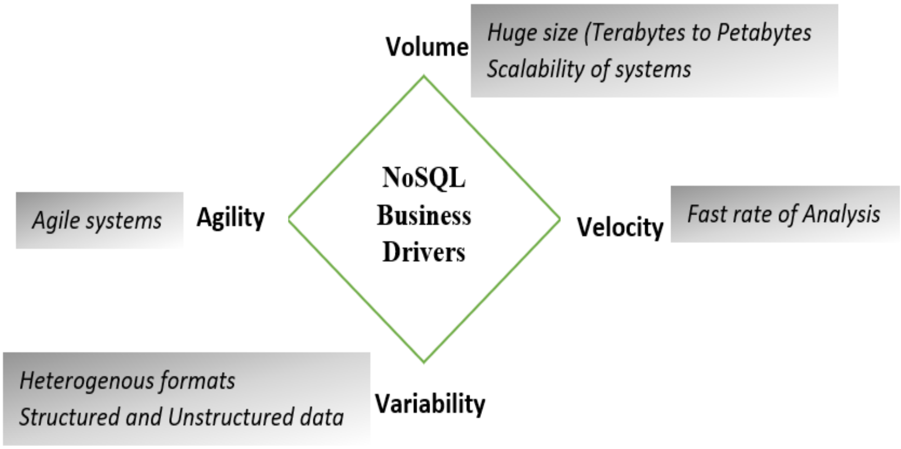
Data Models -
Key-value – Redis, BerkeleyDB, MemcacheDB
Column Oriented – HyperTable, HBase
Document Oriented – MongoDB, Terrastore

5.4 NOSQL BUSINESS DRIVERS

Today, we are living in an age of rampant data growth. As the size of data grows and source of data creation becomes increasingly diverse, we are faced with following challenges:

- Efficiently storing and accessing large amounts of diverse data is becoming difficult. Fault tolerance and backups make things even more complex.
- Manipulating large datasets involves running immensely parallel processes. Recovering the system gracefully from failures and providing results in a reasonably short period of time is complex.
- Managing the continuously evolving schema and metadata for semi-structured and unstructured data produced by diverse sources is another complex issue to deal with.

In today's world many organizations which supported single-CPU relational data models are rapidly making changes in their businesses based on the data they receive. The organizations realised that the way and means of storing and retrieving large amounts of data need newer approaches beyond their current techniques. NoSQL and related big data solutions were the first step in that direction. The demands of Volume, Velocity, Variability and Agility play a key role in developing cracks in the popularity of single-processor relational systems and emergence of NoSQL technologies.



- **Volume:** We have seen the growth in volume of data from bits to bytes to gigabytes to zettabytes. Traditional relational database infrastructure cannot cope up with these huge volumes of data. Just consider Facebook, since 2016, there are over 2 trillion posts and 270 billion photos uploaded. Enormous volume of data plays an important role for pushing the organizations to look for alternatives to their current RDBMS setup for querying big data. In earlier days, performance concerns were solved by using faster processors. But with the increase in volume of data produced daily, there was a need for horizontal scaling. So eventually the organizations moved to parallel processing from serial processing. In parallel processing data problems are split into separate paths and sent to separate processors to divide and conquer the load.

- **Velocity:** At what speed is new data generated? Rate at which this data is generated is much faster. We have moved from the days of batch processing to real time processing. Single processor RDBMS systems cannot meet the demands of real-time inputs and online queries made by users. When single-processor RDBMS is used as backend for processing online queries, the random bursts in online traffic slows down the response for every user. So, Velocity plays a major role in NOSQL movement.

- **Variability:** How diverse are different types of data? Data now is extremely diverse. Variety deals with wide range of data types and sources of data. So, rigid database schema imposed by RDBMS cannot deal with such heterogenous type of data.

- **Agility:** Traditional databases require a schema before writing data. Moreover, time is needed to get the data into the database and this process cannot be considered as agile. Today, the technologies focus more on agility meaning how fast can you extract value from mountains of data and how quickly can you translate that information into action.

5.5 ADVANTAGES OF NOSQL

Advantages of NoSQL are as follows:

1. **Scalability:** NoSQL technology was designed in Internet and Cloud computing era. So, it implements scale-out architecture. Here scalability is achieved by spreading the data and the load to a large cluster of computers. New computers and storage space can be easily added to the cluster. Whenever data volume increases or traffic grows scalability can be easily achieved in NoSQL. It allows distribution of database across 100+ nodes in multiple data centres. It can sustain over 100,000+ database read and write operations per second. It allows storage of 10,000 lakh documents in the database. In addition, many NoSQL databases can be easily upgraded or changed with zero downtime.

2. **No predefined schema:** Relational databases have predefined schema. To use RDBMS, a data model must be designed and data has to be transformed and then loaded into the database. NoSQL databases are gaining popularity as they allow the data to be stored in ways that are easier to understand. Fewer transformations are required when the data is stored and later retrieved for usage. Moreover, varied forms of data – structured, semi-structured and unstructured can be easily stored and retrieved. As NoSQL databases are flexible, they can easily adapt to newer forms of data. This also helps the developers who find it easier to develop various types of applications using data in native formats.

3. **Economical to use:** NoSQL databases are relatively cheaper to install and manage as compared to traditional datastores. It provides all the benefits of scale, high availability, fault tolerance at lower operational costs.

4. **Distributed and fault tolerant architecture:** In NoSQL, data can be replicated to multiple nodes and can be partitioned. Sharding allows different pieces of data to be distributed across multiple servers. NoSQL databases support auto-sharding. It means that they can natively and automatically distribute data across multiple servers without requiring the application to be even aware of the composition of the server pool. Servers can be added or removed without any application downtime. It means that data and query load are automatically balanced across servers in case a server becomes non-functional. No application disruption is experienced as non-functional server is quickly and transparently replaced. NoSQL also allows replication of data. Here, multiple copies of data are stored across the cluster and data centres. This ensures high availability and fault tolerance.

5.6 SHARDING AND REPLICATION

As data generated and stored withing an organization increases, there may come a point where the current infrastructure becomes inadequate and some mechanism to break/divide the data into reasonable chunks is needed. Organizations may use auto-sharding which includes breaking the database into chunks called as shards and spreading the shards across a number of distributed servers. Each shard is an independent database and collectively they constitute a logical database. Data shards may also be replicated for reasons of reliability and load balancing. On older systems this might mean taking the system down for few hours while the database is reconfigured manually and data is copied from old system onto the new system. But NoSQL systems do all this automatically (e.g., MongoDB). Many NoSQL databases provide automatic partitioning and balancing of data among nodes. Sharding is highly automated process in both big data and fault tolerant systems.

The most two important advantages of Sharding are as follows:

- Sharding reduces the amount of data that each node needs to store and manage. E.g., if the dataset was 2TB in size and we distribute this over 4 shards then each shard will have to store only 512 GB data. As the cluster size increases, the amount of data that each shard has to store and manage will decrease.

- Sharding also reduces the number of operations that each node has to handle. E.g., if we have to perform write operation, then the application has to access only that shard which stores the data related to the write operation.

As the number of servers grows, the probability of any one server being down remains the same. So, you may decide to replicate the data to a backup or mirrored system. Replication Factor means the number of copies of the given dataset stored across the network. In this case, when there are any changes to the master copy of the data, you must also update the backup copies immediately. It means that the backup copies must remain in sync with the master copy. So, you must have a method of data replication. Syncing these databases can reduce the system performance.

5.7 NOSQL DATA ARCHITECTURE PATTERNS

Types of NoSQL databases/data stores:

NoSQL applications use a variety of databases. Each NoSQL type of data store has unique attributes and uses.

1. **Key-value store:** It maintains a big hash table of keys and values. It is a simple data storage system that uses a key to access a value. The key of a key/value pair is a unique value in the set and can be easily looked up to access the data. Within a key-value pair, there are two related data elements. The first is a constant used to define the dataset. The other is a value which is a variable belonging to the dataset.

For e.g., a mobile phone can be the key, while its colour, model, brand can be the values.

Another example can be of the bank. A bank can use a database to store key-value pairs that contain their customers information. Customer's name can be the key while his/her account number, account type, branch, balance can be the values.

Key/value pairs can be distributed and held in a cluster of nodes. Its typical usage includes Image stores, session management for web applications, user sessions for massive multi-player online games, online shopping carts etc.

A powerful key/value store is Oracle's Berkeley DB which is a pure storage engine where both key and value are an array of bytes.

Another type of key/value store in regular use is a cache. A cache provides an in-memory snapshot of the most frequently used data in an application. A popular open source, high performance object caching system used in web applications is Memcached.

Another popular open source in-memory NoSQL key/value store is Redis (REmote DIctionary Server) which is primarily used as an application cache or quick-response database. Microsoft uses Redis to power MSN portal that gets 2 billion hits with a latency of less than 10ms on peak days.

Amazon's Dynamo is also a key/value pair which emphasize availability over consistency in distributed deployments. It forms the backbone for Amazon's distributed fault-tolerant and highly available system. Cassandra (used by Facebook, Reddit, Twitter, Spotify, eBay), Riak and Voldemart (used by LinkedIn) provide open-source Amazon Dynamo capabilities.

2. **Document Databases:** It maintains data in collections constituted of documents. Here hierarchical data structures are directly stored in the database. Document databases treat a document as a whole and do not split a document into its constituent name/value pairs. The document is stored in JSON or XML formats.

For example: Sample document in Document database

```
{ “Name” : “ABC”,
  “Address” : “Indore”,
  “Email” : abc@xyz.com,
  “Contact” : “98765”
}
```

Document databases are simple and scalable, thus making them useful for mobile applications that require fast iterations. These data models are also used in video streaming platforms, blogs and similar services.

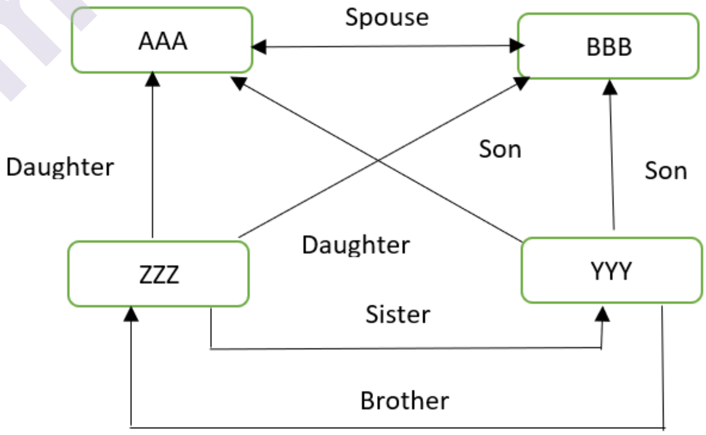
Most popular open-source document databases are MongoDB (used by GitHub), Amazon DocumentDB and Apache CouchDB (used by Apple, LinkedIn).

3. **Graph database:** It is also called as Network database. These datastores are based on graph theory. A graph database can have ACID properties. A graph stores data in nodes. It is designed to identify and work with the connections between data points. The fundamental data model of a graph database is very simple – nodes connected by edges. The entity is stored as a node with the relationship as edges. An edge gives a relationship between nodes. Every node and edge have a unique identifier.

The simplest possible graph is a single node. This would be a record with named values called properties/attributes. Theoretically there is no upper limit on the number of properties in a node, but practically we distribute the data into multiple nodes, organized with explicit relationships. When we store a graph structure in RDBMS, we can actually store single type of relation and adding another relation type requires changing the schema structure. A graph database is schema free and we can scale up to any level by adding a different type of entities and relations (nodes and edges).

It is typically used in mining data about customers on social networks, fraud detection, product preferences, eligibility rules etc.

Two popular Graph databases are Neo4j (ACID-compliant graph database used by Walmart to improve the speed and effectiveness of their online recommendation platform.) and FlockDB (used in Twitter – stores the adjacency lists for followers on Twitter). FlockDB is simply nodes and edges with no mechanism for additional attributes whereas Neo4j allows you to attach Java objects as attributes to nodes and edges in a schema-free way.



4. **Column store:** It's a sparse matrix system that uses a row and a column as keys. Here each storage block has data from only one column. It avoids consuming space when storing nulls by simply not storing a column when a value does not exist for a column. A column family is a collection of rows, which contains any number of columns

for each row. The row must be unique within a column family but the same row can be used in another column family.

CustomerID	Column Family : Identity
100	First Name : AAA Last Name : ZZZ
101	First Name : BBB Middle Name : KKK Last Name : YYY
102	Title : Dr. First Name : CCC
CustomerID	Column Family : Contact Info
100	Mobile : 555555 Email : a@eg.com
102	Email : b@asia.com
103	Mobile : 12345

The benefit of storing data in the column datastore is fast searching and accessing of the data. These type of datastores are typically used in data warehousing as they handle analytical queries very efficiently.

HBase (used by Facebook, Yahoo!) is a popular open-source, sorted ordered column family store. Data stored in HBase can be manipulated using MapReduce infrastructure. MapReduce tools can easily use HBase as the source and/or sink of the data.

Following table summarises comparison among different NoSQL databases and RDBMS based on various important features:

	Performance	Scalability	Flexibility	Complexity	Functionality
Key-Value Data Store	High	High	High	None	Variable
Column Oriented Data Store	High	High	Moderate	Low	Minimal
Document Data Store	High	Variable	High	Low	Variable
Graph Data Store	Variable	Variable	High	High	Graph Theory
Relational Databases	Variable	Variable	Low	Moderate	Relational Algebra

Following table summarises popular NoSQL products and its usage:

	Key-Value Data store	Column Oriented Data store	Document Data Store	Graph Data Store
Popular Products	Oracle's Berkley DB, Redis, Riak, Amazon's Dynamo	HBase, HyperTable	MongoDB, Amazon DocumentDB, Apache CouchDB	Neo4j, FlockDB, InfiniteGrpah
Usage	Shopping carts, web user data analysis	Analyze huge web user actions, sensor feeds	Real-time analytics, logging, document archive management	Network modelling, recommendation system
Mostly used by	Amazon, LinkedIn	Facebook, Twitter, eBay, Netflix	GitHub, Apple	Wallmart

5.8 USE OF NOSQL IN INDUSTRY

NoSQL is used in variety of applications in industries. Some uses are listed below:

1. **Session Management:** NoSQL is used for storing web application session information which is very large in size. Such session data is of unstructured format so it is advisable to store it in a schema-less document.

2. **Storing User Profile:** It is necessary to store user's profile to enable online transactions, to understand user preferences and carry out user authentication. In recent years users of web and mobile applications have grown two-fold so to handle such deluge NoSQL can be easily used to increase the capacity by adding servers which makes scaling cost effective.

3. **Content and Metadata store:** Many organizations require storage for storing huge amount of digital content, articles, e-books etc. in order to merge a variety of learning tools in a single platform. The content-based applications require frequent access to metadata so it should have minimum response time. NoSQL provides flexibility to store different types of contents and also provides faster access to data.

4. **Mobile Applications:** Users of smartphones are increasing at an enormous pace. Mobile applications face issues related to growth and volume. With relational databases, it becomes really difficult to expand as number of users increase. NoSQL allows us to expand without worries. Moreover, as NoSQL allows data to be stored in

schema-less fashion, mobile application developers can easily update the applications without doing major modifications to the database.

5. **Internet of Things:** All the devices which are connected to the internet like smartphones, laptops, tablets, home appliances, hospital systems, car systems etc produce large amount of data which is heterogenous in nature. Traditional data stores are not adequate to deal with this deluge. NoSQL allows organizations to expand concurrent access to data from billions of devices and systems which are connected to internet, meeting the required performance criteria.

6. **E-Commerce:** Most of the E-Commerce companies use NoSQL for storing huge amount of user data and user's preferences. NoSQL also allows the companies to deal with peak traffic of user request and queries.

7. **Social Gaming:** Users of social gaming are increasing day-by-day. Data intensive gaming applications require a database system which can be scaled to incorporate the growing number of users. NoSQL suits this requirement.

8. **Ad targeting:** For displaying ads or lucrative offers on the webpages, the web portal gathers behavioural and demographic characteristics of its users. Web Portals should decide which group of users to target, where to place ads on the webpage so that it gets maximum clicks. A NoSQL database allows ad companies to track user details and makes intelligent decisions thereby increasing the probability of clicks.

5.9 NOSQL AND BIG DATA

Big Data refers to datasets whose size (volume), complexity (variability) and pace of growth (velocity) makes it difficult to be captured, stored, managed, processed and analysed by using the traditional and conventional tools and technologies like RDBMS and data processing applications. The growth of the data at enormous speed is just one facet of the big data problem, the other is the necessity to store and manage not only the structured data but images, videos, files and all sort of semi-structured and unstructured data. Big Data needs a whole set of new generation of technologies which are designed to query and analyse large volumes of heterogenous data. The problems of large datasets that need rapid analysis won't go away in the near future.

"Big Data is data whose scale, distribution, diversity, and timeliness require the use of new technical architectures and analytics to enable insights that unlock new sources of business value."

Due to its size or structure, Big Data cannot be efficiently analysed using only traditional databases or methods. Big Data problems require new tools and technologies to store, manage, and realize the business benefit.

Using a NoSQL solution to solve your big data problems gives you some unique ways to handle and manage your big data.

Big data problems force you to move away from a single processor infrastructure towards a more complex world of distributed computing. Distributed databases are more complex than a single processor system and has its own challenges.

5.10 UNDERSTANDING BIG DATA PROBLEMS

Consider following use cases:

- **Bulk image processing:** There are many organizations which regularly generate, store and process images. They also perform various operations on the images like image enhancement, image upscaling, photo stitching etc. Medical imaging systems like MRIs or CAT scans generate raw image data which in turn has to converted into formats which are understandable to doctors and patients. If we use custom imaging hardware then it is not at all cost effective. So, it is advised to rent a large number of processors on the cloud whenever needed.

- **Public web page data:** Today there are millions of webpages are easily accessible to all the internet users. These webpages display news stories, product reviews, blogs etc. Not all the information displayed on these webpages is authentic. Many a times, competitors create such fake webpages to affect the business of an organization.

- **Remote sensor data:** With IoT devices being used in our daily lives, these small sensors can easily track almost everything about us. Sensor devices installed in vehicles can track location, speed, distance covered, fuel consumption thereby informing the insurance company about your driving habits. Now we can also warn about traffic jams in real time and suggest alternative routes to reach the destination using road sensors. We can track when the garbage bin is full using sensors. Sensors track moisture levels in the garden and come up with suggestions regarding best plants for the garden and their watering plan.

- **Event log data:** Clickstream data is an information trail a user leaves behind while visiting a website. This information trail contains data elements such as a date and time stamp, the visitor’s IP address, the URLs of the pages visited, and a user ID that uniquely identifies the user. All these details are stored in quasi-structured website log files. Systems also create logs when a user attempts login or sends an email.

- **Mobile phone data:** Companies are finding innovative ways to access your mobile phone data (even though it violates the privacy of the user) to get ahead of their competitors. Smartphones provide another rich source of data. In addition to messaging and basic phone usage, they store and transmit data about Internet usage, SMS usage, and real-time location. This metadata can be used for analysing traffic patterns by scanning the density of smartphones in locations to track the speed of cars or the relative traffic congestion on busy roads.

- **Social media data:** Social media networks like Facebook, Twitter, LinkedIn etc. provide a continuous real time data stream. For example, a small update on your social media account bombardes you with related information or ads. Facebook can also construct social graphs to analyse which users are connected to each other as an interconnected network. Way back in 2013, Facebook released a feature called "Graph Search," enabling users and developers to search social graphs for people with similar interests, hobbies, and shared locations.

- **Gaming data:** Consider someone playing an online video game through a PC, game console, or smartphone. In this case, the video game provider captures data about the skill and levels attained by the player. Intelligent systems monitor and log how and when the user plays the game. As a consequence, the game provider can fine-tune the difficulty of the game, suggest other related games that would most likely interest the user, based on the user's age, gender, and interests. This information may get stored locally or uploaded to the game provider's cloud to analyse the gaming habits and opportunities for increasing the sale.

After analysing the above use cases, we can conclude that some use cases focus on efficient and reliable data transformations at scale. While some use cases like event log data and game data need to store their data directly into structures that can be easily queried and analysed.

NoSQL big data solutions are efficient and scale linearly as data volume increases. NoSQL big data solutions use distributed computing while considering latency between systems and node failures. NoSQL systems also isolate the developers from the complexities of distributed computing.

5.11 EXERCISES

1. Explain in brief the various features of NoSQL.
2. What is CAP theorem? How it is different from ACID?
3. Explain the difference between RDBMS and NoSQL.
4. What are the advantages of NoSQL over traditional RDBMS?
5. When to use NoSQL datastore instead of traditional RDBMS?
6. Explain in detail what was the driving force behind designing NoSQL database?
7. Explain the difference between scaling horizontally and vertically in databases.
8. Explain the term Scalability.
9. What are the different types of NoSQL datastores? Explain each one in short.

10. Explain in detail Column-Oriented Datastore.

11. Explain in detail Graph database.

12. How does Document Datastore differ from Column Oriented datastore?

13. How does Document Datastore differ from Key-Value datastore?

14. Explain what is Sharding along with its advantages.

15. What are the various applications of NoSQL in industry?

16. Explain how companies are using NoSQL to target the users/potential customers on social media or web pages with their products?

17. Explain how Internet of Things industry is using NoSQL?

18. What to do understand by Bigdata?

19. Explain how Big Data is affecting the companies in the way they do their business.

20. What are the various Bigdata problems?

21. Explain the Big Data use cases related to social media data and mobile data.

22. Explain the Big Data use cases related to gaming data and web page data.

23. How NoSQL is used to deal with Bigdata?

24. Explain how Replication is used as one of the ways to deal with Bigdata in NoSQL?

5.12 ADDITIONAL REFERENCES

1. 'Making Sense of NoSQL' by Dan McCreary, Manning Publications

2. 'Joe Celko's Complete Guide to NoSQL' by Joe Celko, Newnes Publications

3. www.w3resource.com/mongodb/nosql.php

4. www.mongodb.com/nosql-explained

5. www.javatpoint.com/nosql-databases

6. www.geeksforgeeks.org/use-of-nosql-in-industry

*****

6

USING NOSQL TO MANAGE BIG DATA AND HBASE

Unit Structure

6.0 OBJECTIVES

This chapter would make you understand the following concepts:

- Relation between NoSQL and Big Data landscape
- Understanding big data problems with the help of various use cases
- Popular ways in which NoSQL handles big data
- Comparing Shared Memory and Shared Disk with Shared Nothing Architecture
- Distribution model – Master Slave vs Peer to Peer
- Introduction to HBase as one of the popular NoSQL databases along with its features and advantages.
- Information on some companies along with their use cases that are using HBase
- HBase data model and its key components
- Major components of HBase Architecture and information on Compaction
- Read Write Mechanism in HBase

6.1 HOW NOSQL HANDLES BIG DATA PROBLEMS?

Following are the four most popular ways in which NoSQL handles big data:

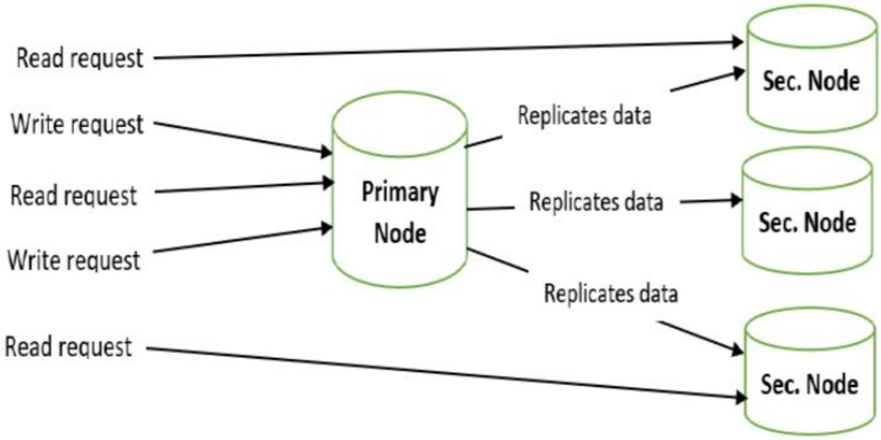
1. **Moving queries to the data and not vice versa:** Most of the NoSQL systems depend on cheap commodity processors that hold a subset of data on their shared nothing drives. Traditional database systems cannot distribute queries and aggregate query results, whereas NoSQL sends a query to each node instead of transferring large datasets to the central processor. In traditional database systems data movement of large datasets results in slower processing of the queries. In NoSQL as the dataset remains on the node itself, only query and its final result moves over the network thereby resulting in speedy results.

2. **Using hash rings to evenly distribute data on a cluster:** Distributed database systems have to find a consistent way of assigning a document to a processing node. Hash ring is one of such technique. Hash rings can consistently determine how to assign a document to a specific processor. Hash rings use the leading bits of a document's hash value to determine the node to which the document is assigned to. With this scheme, any node in a cluster can easily determine the node on which data resides and new assignment rules to be adapted as new nodes are added to the cluster. Key space management refers to the methods in which keys are partitioned into ranges and different key ranges are then assigned to specific nodes.

For e.g., Consider a key allocation strategy among a set of nodes which uses modulo function. Suppose 50 keys are distributed among 7 nodes in following way: key with value 85 goes to node 1 because 85 modulo 7 is 1 and key with value 18 goes to node 4 because 18 modulo 7 is 4, and so on. This strategy works successfully until the number of nodes changes, that is, newer ones are added or existing ones are removed. When the number of nodes changes, the modulo function applied to the existing keys produces a different output and leads to rearrangement of the keys among the nodes.

Hash ring scheme can also be expanded to include replication of dataset on multiple nodes. When a new data item is created then the hash ring scheme can determine the primary and secondary node on which the item is to stored. (Secondary node on which the item is replicated.) In this case, when the primary node fails, then the system can determine node on which the replicated data item is stored.

3. **Using replication to scale up read operations:** NoSQL uses replication to store backup copies of datasets on multiple nodes. Load balancers distribute queries to the correct database server. Replicating datasets can actually speed up read operations in NoSQL systems. All the read requests can be directed to any node either primary node or secondary node (where replica of the dataset is stored). All the write requests are directed only to the primary node which updates the dataset and immediately sends the updates to the secondary nodes. You must be concerned about the time lag between the write to the primary node and arrival of the update to the secondary node. If the user reads the data from the secondary node before the update arrives then it will lead to data consistency issues.



The solution for this issue is to allow read operations to the same write nodes once a write has been done. This can be added to the state management system at the application layer. Fast read and write consistency must dealt at the application layer.

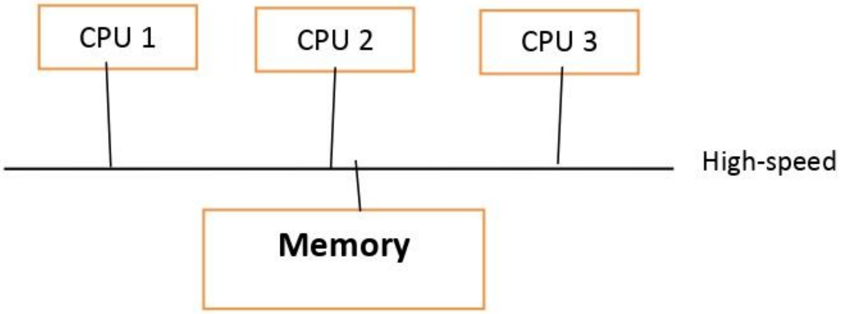
4. Even distribution of queries to data nodes – The approach of NoSQL systems is moving the query to the dataset on the node rather than moving the dataset to the query. In NoSQL system, database server is responsible for moving the query and distribution of the query and waiting for all nodes to respond is the sole responsibility of the database. In this approach a single query from the client is distributed to distinct servers and then the results returned from various servers is combined and send to the client giving him an impression that result is coming from a single server.

6.2 ANALYSING BIG DATA USING SHARED NOTHING ARCHITECTURE

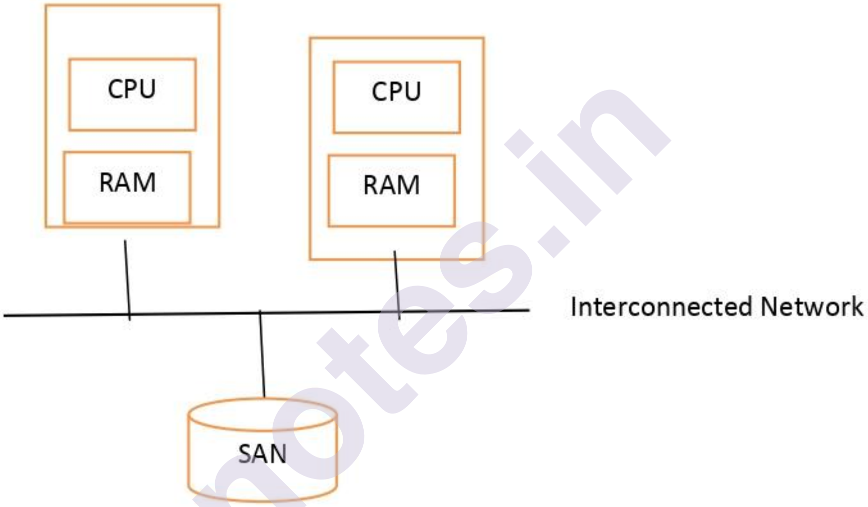
There are three ways in which resources can be shared between computer systems. They are as follows:

- Shared RAM (shared memory)
- Shared Disk (shared storage)
- Shared Nothing

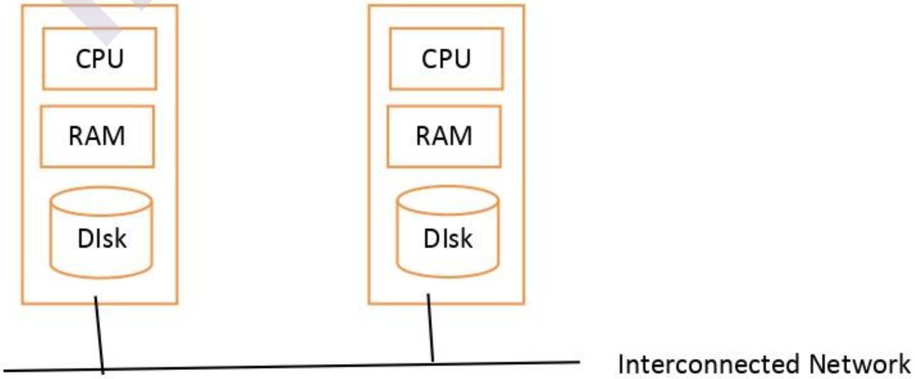
Shared RAM (shared memory) architecture: Here multiple CPUs access a single shared RAM over a high-speed bus. This system is used for large graph traversal. This architecture offers simplicity and load balancing as it includes point-to-point connections between devices and main memory.



Shared Disk (shared storage) architecture: In this type of architecture, processors have independent RAM but they share disk using a storage area network (SAN). The shared disk architecture is best for systems where data partitioning is not on option.



Shared Nothing architecture: This is the most cost-effective architecture used to solve big data problems. Shared nothing architecture is the one where neither memory nor peripheral storage is shared among processors.



Advantages of Shared Nothing Architecture:

1. **Scalability:** Unlimited scalability is one of the best features of shared nothing architecture. Scaling up your application won’t disrupt the entire system or lead to resource contention as here independent nodes do not share resources.

2. **Performance:** Here, it is easier to achieve higher consistent performance as it delegates load to different parts of a system. In Shared Nothing architecture there is reduction in resource contention which eventually leads to higher performance and lower latency in processing individual requests.

3. **Fault Tolerance:** If a node fails, it does not affect the functionality of others as each node is self-reliant. Node failure definitely affects performance but it does not affect the overall behaviour of the application as a whole.

4. **System downtime:** There is no need to shut down the system when individual nodes are updated. In shared nothing architecture upgradation of one node does not affect the effectiveness of other nodes. Moreover, as replicated datasets already reside on multiple nodes, there is zero system downtime in case of any kind of failure.

Though Shared Nothing architecture has its advantages, having dedicated resources like individual processor, memory and disk, increases the costs of setting up the system.

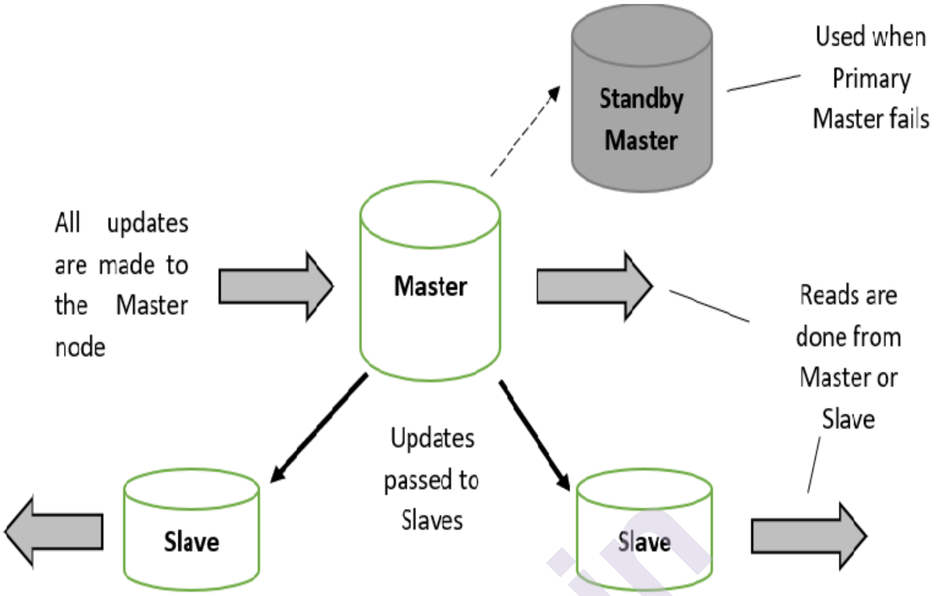
6.3 DISTRIBUTION MODELS – MASTER - SLAVE VS PEER-TO-PEER

The main attraction of NoSQL has been its ability to run databases on large clusters. Distributing datasets over clusters introduces complexity but many a times its benefits are compelling. The responsibility of processing data whenever a request is made There are two main distribution models – Master-Slave model and Peer-to-Peer model.

- **Master:** Slave model: In this model, one node is in charge, known as Master node or Primary node and the other nodes are known as Slave nodes or Secondary nodes. Master node is responsible for managing the cluster. Here, all incoming database requests of read or write are sent to a single master node and redistributed from there. Master node keeps a database of all other slave nodes in the cluster and the rules for distributing requests to each node. This node requires specialized hardware such as RAID drives to avoid crashing.

Master: Slave replication helps to scale when large number of read operations are performed on the dataset. Read requests are routed to the slaves. More read requests can be handled by adding more slave nodes to the cluster. Master should process the updates and pass those updates without any delay to all the slave nodes. So, it can be argued that this is not a great model for datasets with heavy write traffic.

Another advantage of this distribution model is Read Resilience which means even if a master fails, slaves can still handle read requests. This is useful when dataset has heavy read traffic.

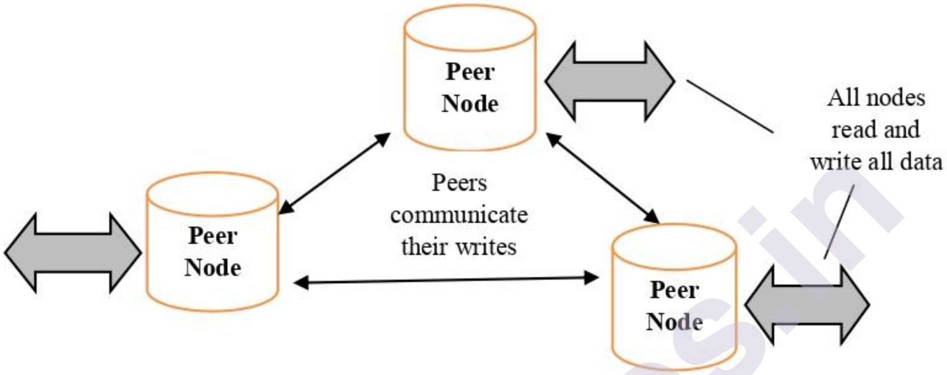

Standby Master
Used when Primary Master fails
All updates are made to the Master node
Master
Reads are done from Master or Slave
Updates passed to Slaves
Slave
Slave

Failure of master node affects the write operations until either the master is restored or a new master is chosen. Sometimes, one of the slaves is appointed as master to speed up the recovery after a failure. This can be done as slaves are replicates of the master. The process of appointing a master in a cluster can be done manually or automatically. In manual process, during the configuration of the cluster, one node is selected and appointed as Master manually. In automatic process, once the cluster is configured, the nodes elect one of them as the Master node. In this scheme, in case of a failure, new master is automatically chosen thereby reducing the system downtime.

Another solution is having a Standby Master. This standby master node is continually updated by the master node. It's challenging to test this standby master node without risking the wellbeing of the cluster. For high-availability operations, it's very important to ensure that the standby master takes over the master node in case of failure.

- **Peer – to – Peer model – Master :** Slave model efficiently deals with heavy read traffic but may face issues with heavy write traffic. Master – Slave model can easily deal with the failure of slave nodes but faces problems in dealing with the failure of master node. In master – slave model, the master node is the bottleneck and root cause of the issues. So, in Peer – to – Peer model these problems are solved by not having a master node. All the nodes of the cluster have equal weight, and each one stores all the information about the cluster. All the nodes are treated as Peers and can accept write requests. Loss of any of the peer node does not stop the client from accessing the dataset, as other nodes continue to function. Peer nodes can be easily added to the cluster to improve the performance. But this model is a bit complex and experiences communication overhead. The most prominent issue here is consistency. All the nodes need to be kept updated and consistent.

For e.g., when there are two write operations going on at the same time to two different peers, there is a risk the both these write operations are updating the same record at the same time on different peers. This leads to inconsistent data. This is known as write-write conflict. So, we need to ensure that whenever a write operation occurs, peers coordinate to avoid conflicts. Nodes should coordinate to synchronize their copies of data. A policy has to be made to merge all the inconsistent writes. All this coordination between the peers increases the network traffic leading to increase in communication overhead.


Peer Node
Peer Node
Peer Node
Peers communicate their writes
All nodes read and write all data

Master – Slave model reduces the possibility of update conflicts whereas Peer – to – Peer model avoids loading all the write operations onto a single point of failure which is the master node.

6.4 HBASE OVERVIEW

The rise and popularity of big data resulted in NoSQL databases. HBase is one of the NoSQL databases built on top of Hadoop as a scalable data store and so the name HBase actually means Hadoop database. HBase is an opensource NoSQL database which is a Java implementation of Google's BigTable. (BigTable is a distributed storage system for managing structured data that is designed to scale to a very enormous volume across thousands of commodity servers).

HBase was created in year 2007 by Powerset (acquired by Microsoft in 2008). HBase is a part of Apache Software Foundation umbrella, so it is also called as Apache HBase. As HBase is modeled on Google's BigTable, it provides distributed data storage system. It implements storage architecture from column-oriented databases and data access design from the key-value store databases where a key-based access to a specific cell of data is provided. Column-oriented databases store data grouped by columns and column values are stored contiguously on a storage device. This results in reduced I/O. This kind of architecture is highly effective when dealing with large datasets where all the columns are not needed for analytical queries. Moreover, HBase can store both structured, semi-structured and unstructured data using a cluster of commodity servers. So, it can store tweets, parsed log files, a catalog of all the products along with customer reviews etc. It has an ability to scale horizontally as more devices are added to the cluster. Each node in the cluster provides storage, cache and computation services. HBase is extensively used in applications where there is heavy read/write traffic to large datasets.

HBase is a distributed, persistent, strictly consistent storage system with excellent read/write performance, and makes efficient use of disk space with the help of pluggable compression algorithms which can be selected based on the nature of data in specific column families.

Key Features of HBase are as follows:

- Horizontally scalable
- Ability to store large datasets
- Strong consistency
- Real-time lookups
- Automatic load balancing of datasets
- Fault-tolerant capability
- Easy and efficient Java API available for clients
- Supports parallel processing, HDFS and MapReduce

Benefits of HBase over other NoSQL databases:

- It gives a direct access to Hadoop scale storage due to which read-write operation to specific data can be performed quickly.
- It can handle volume, complexity and variety of Hadoop database easily as it provides transactional platform for running high-scale applications.
- Hadoop's ecosystem chooses HBase over other NoSQL databases as it is consistent and gives high quality performance.
- It is schema independent so it can be used in various applications.
- The integration of YARN (also known as MapReduce 2 allocates and manages the resources in Hadoop) and HBase makes it a very good technology for Big Data applications.

Other NoSQL databases like MongoDB, Cassandra, Redis are strong competitors of HBase.

6.5 USE CASES OF HBASE

HBase has proved to be a very powerful tool and a core infrastructure component in various companies like Facebook, Adobe, Twitter, Trend Micro, StumbleUpon etc. As more commercial vendors provide support, users are increasingly showing their confidence for using HBase in critical applications. From storing communications between individuals to communication analytics, HBase is popularly used in big data companies.

Following are some of the companies along with their use cases that are using HBase:

1. **Capturing Incremental data:** Trickled data is often added to the existing storage for further analytics, processing and serving. Data often trickles in from various sources like web crawls, advertisement impression data containing information about which user saw what advertisement for how much time, or time series data produced from recording metrics of different types.

- **Capturing metrics (StumbleUpon's OpenTSDB and Twitter):** Web-based products often have thousands of servers working as backends. These are used for serving traffic, capturing logs, storing data, processing data etc. It is very essential to monitor the health of these servers as well as the software running on these servers. So, the companies require systems that can collect and store metrics of all kinds from different sources.

'StumbleUpon' designed an open-source framework which a company can use to collect metrics of all kinds into a single system. Metrics collected over time can be seen as time-series data. StumbleUpon designed OpenTSDB (which stands for Open Time Series Database), which uses HBase at its core for storing and accessing the metrics. This framework also allows new metrics to added as more features are added to the product. StumbleOpen uses OpenTSDB to monitor all of its infrastructure and software including HBase clusters.

Twitter also uses HBase as a time series database for cluster-wide monitoring / performance data.

- **Capturing user-interaction data (Facebook):** The major challenge is to keep track of activity of millions of people visiting your site. Also, you need to know which site features are most popular. It is very important to determine how many times a particular button the site was clicked. E.g., Like button on Facebook.

Facebook uses HBase counters for counting and storing the 'likes' contributed by users. Facebook built a system called Facebook Insights which is backed by HBase's scalable storage system. The system handles tens of billions of events per day and records thousands of metrics.

- **Telemetry (Mozilla and Trend Micro):** Like metrics, crash reports can be used to gain insights into the quality of software and plan further modifications to the software. HBase captures and stores crash reports that are generated from software crashes at user's end.

Mozilla Foundation (their popular products are Firefox web browser and Thunderbird email client) collects bug reports when one of their product crashes on client’s device. These crash reports/bug reports are collected using a system called as Socorro whose data storage and analytics are built on HBase. Data analysed from these collected crash reports helped Mozilla to develop more bug-free versions of their products.

Trend Micro provides threat management and internet security services to corporate clients. Trend Micro uses HBase to collect billions of records every day and analyse log activity. HBase also provides row-level updates and flexible schema so that the data can get evolve over time.

- **Advertisement impressions and Clickstreams:** Online advertisements are now a major source of revenue for various companies. These advertisements target users and attract them to their products. This kind of targeting requires detailed analysis of user profile and his preferences. The advertisement to be displayed is then chosen based on this analysis. Correct analysis leads to better targeting leading to increased revenue. HBase has been successfully used to capture such raw clickstreams and user-interaction data incrementally and then process it using different mechanisms like MapReduce.

2. **Content Serving:** Many scalable content management solutions are using HBase for storing the data. Today we have various devices like laptop, TV screen, mobile, tablets etc. where the user wants to display the content from the Internet. This leads to a requirement where the same content has to be displayed in different formats. Moreover, user not only consumes content, but he produces a large volume of content at high pace using social media posts, blogs, tweets, image uploads etc. HBase provides the perfect backend solution for handling large volume of content/data. Lily Content Management System uses HBase as its backend and open-source framework Solr for storing and serving content.

- **URL Shorteners:** URL shorteners also known as Link shorteners convert your long URLs into a short and precise form. StumbleUpon’s URL shortener tool su.pr uses HBase to shorten URLS and to store thousands of short URLs along with their mapping to the long URLs.

- **Serving User Models:** We know that the metadata which is collected also plays a very important in analysis. The data stored in HBase may not be directly used by the users, instead it can be used to make analytical decisions about what should be served to the user. User profile data stored using HBase is being used to decide what advertisement to serve to the user, to add context to user interactions, to decide pricing and discount offers in real time when users do online shopping, search results on a search engine etc.

3. **Information Exchange (Facebook and Meetup):** Today millions of people communicate in groups or individually over social networks. Social networks don’t just allow people to communicate with each other but also allow them to look at the history of all their

communications with others. Innovations in big data systems allow the social network companies to have cheap storage.

Facebook messages feature is backed by HBase. All the messages that users write and read over Facebook are stored using HBase. Here, HBase provides high write throughput, large storage tables, and strong consistency. Billions of messages are exchanged every day on Facebook resulting in about 75 billion operations per day. During peak hours on Facebook, there can be around 1.5 million operations per second on HBase clusters. Facebook adds more than 250TB of new data to HBase clusters every month. This is considered to be one of the largest HBase deployment both in terms of number of users the application serves and the number of servers.

Meetup uses HBase for site-wide, real-time activity feed system for all its users and groups. Group activity is written directly to HBase and indexed per user, with the user's custom feed served directly from HBase for incoming requests.

6.6 HBASE DATA MODEL

A typical relational database contains rows and columns. Relational databases are row-oriented which stores table records in a sequence of rows. Whereas column-oriented databases store table records in a sequence of columns. HBase is a column-oriented NoSQL database. HBase models data in a little different way than the typical RDBMS.

HBase organizes data into tables. Data is stored according to its rows, within a table.

A row in HBase stores different number of columns and datatypes, making it ideal for storing semi-structured data. Both logical and physical schema are affected as HBase stores semi-structured data. Rows here are identified by a unique row key similar to Primary key in RDBMS. Rowkeys do not have a data type and are treated as a byte [ ].

Columns in HBase are arranged into column families. No limit is placed on the number of columns which can be grouped together forming a single column family. This column family is a part of the data definition statement used to create HBase table. Data within a column family is identified using a column qualifier. Column qualifiers do not have a datatype and are treated as a byte [ ]. Each column may have multiple versions, with each distinct value contained in a separate cell.

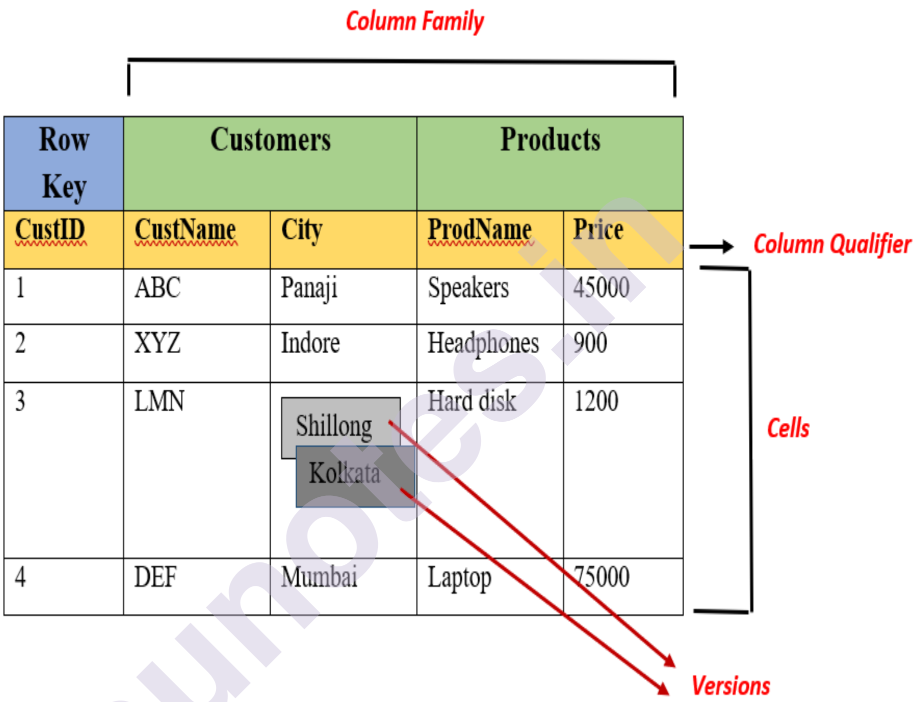
Cell is the placeholder for the column value. Cells are just uninterpreted byte arrays which the user should know how to handle. A combination of rowkey, column family, column qualifier and version uniquely identify a cell. Each cell is either timestamped implicitly by the system or can be timestamped explicitly by the user. This allows us to save multiple versions of a value as it changes over time. The user can specify the number of versions of a value to be kept. The default number of cell versions are three. The API automatically picks up the most current value of each cell.

So, we can express the access to the data as:

(Table, Rowkey, Column Family, Column, Version) Value

So, entities like Table, Row, Rowkey, Column family, Column qualifier and Cell form the foundation of HBase data model. They remain exposed to the normal user via the logical view presented by the API.

Below is an example of HBase table along with its key components:



Row Key	Customers		Products	
CustID	CustName	City	ProdName	Price
1	ABC	Panaji	Speakers	45000
2	XYZ	Indore	Headphones	900
3	LMN	Shillong Kolkata	Hard disk	1200
4	DEF	Mumbai	Laptop	75000

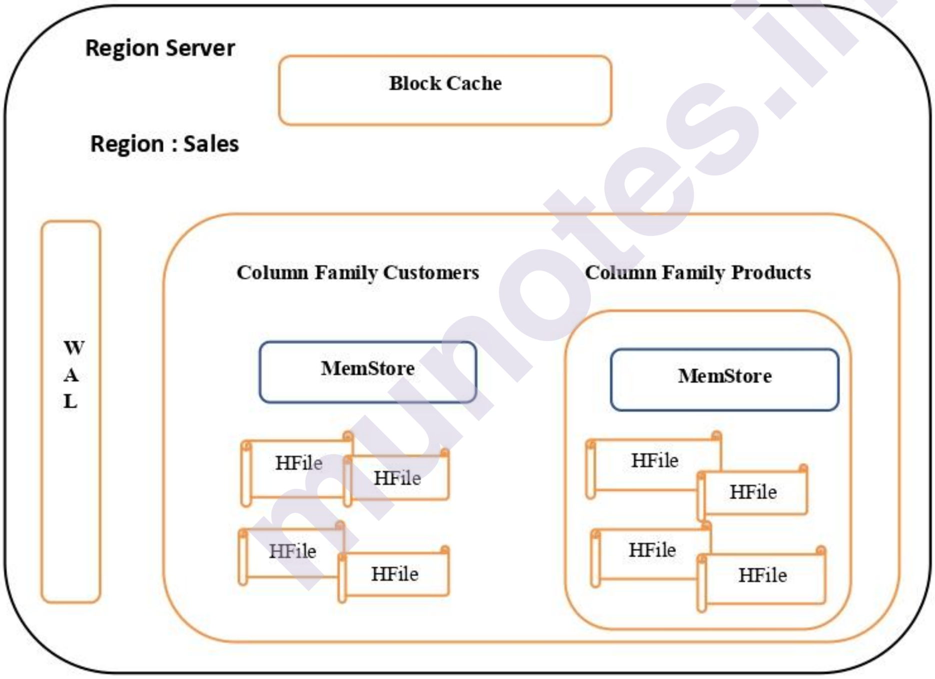
At physical level/storage level, all columns in a column family are stored in a single file called HFile, as key-value pairs. Each column family gets its own set of HFiles on physical storage. So, HBase need not read all the data for a row when performing a read operation. It needs to only retrieve data for the requested column families. Having separate HFiles for each column family allows for isolation among different HFiles. HFile is a binary file and is not in human readable format.

If we add another column family say Orders to the above table then it will result in more HFiles.

Another important concept in HBase is Regions (which are similar to partitions in RDBMS). In HBase tables are stored in regions. It groups the continuous range of rows and stores them together at lower levels in region servers. When a Table becomes too large in size then it is partitioned into multiple regions. These regions are assigned to Region Servers across the cluster. A Region Server is responsible for hosting and managing regions. Mostly each Region Server hosts round about the same number of regions. A Region Server can serve approximately 1000 regions to the client. A maximum size is defined for regions and once the limit is reached, the region is divided into two sections which is known as Region Split.

The Region Server contains the Region Sales which in turn contains two column families namely Customers and Products. Each column family has its in-memory storage and set of HFiles. When write operation is performed then it goes to WAL (Write Ahead Log) and MemStore. HFile is created once the data present in the MemStore is removed to the disk. MemStore holds in-memory modifications to the data. It stores all the incoming data before it is committed to the disk. WAL is a file attached to every Region server. It stores the fresh data that has not yet been committed to the permanent storage. It is used in case of failure to recover the datasets. Block Cache resides in the top of Region Server. It stores frequently read data in the memory.

The below given diagram shows the organization of various data storage level components.



6.7 HBASE ARCHITECTURE

Following are the major components of HBase Architecture:

- **Region Server:** In HBase, regions are sorted range of rows stored continuously. Set of regions are stored on Region Server which is responsible for managing and executing read and write operations on those regions. Region Server (also called as Slave Nodes) is responsible for server data and is collocated in the cluster.
- **HBase Master:** HBase Master (HMaster) is responsible for handling a collection of Region Servers which reside on Data Node. On start-up it assigns regions to Region Servers. In case of recovery or load

balancing it reassigns regions to the Region Servers. It coordinates the HBase cluster and handles the administrative operations. With the help of Zookeeper, it monitors all the instances of the Region Servers in a cluster and performs recovery operations whenever a Region Server goes down. HMaster performs the DDL operations like creation and deletion of tables by providing an interface for the same.

- **Zookeeper:** Zookeeper is the coordinator which helps HMaster in managing the HBase's distributed environment. Zookeeper receives signal at regular intervals from Region Servers and HMaster Server indicating that the servers are alive and functioning. If the Zookeeper fails to receive the signal, it generates server failure notifications and recovery measures are implemented. When a Region Server fails, Zookeeper notifies to the HMaster about the failure. HMaster then allocates the regions of the failed Region Severs to other active Region Servers.

Zookeeper also maintains .META table which helps client to search for any region. The .META table is a special HBase catalog table. The table maintains a list of all the Region Servers in HBase storage system. The table consists of keys and values. A key represents the start key of the region and its id whereas the value represents the path of the Region Server. .META table informs the client of the Region Server path in which a specific region belongs. When the client interacts with the HBase system, the interaction actually starts with Zookeeper and then proceeds to the Region Server serving the region with which the client desires to interact.

Compaction:

Compaction process chooses some HFiles from a region and combines them. During Compaction process, HBase reads the data from the existing HFiles and writes it to new merged HFile. Compaction reduces the storage space occupied. It also reduces the number of disks seeks needed for a read operation. Restricting the number of HFiles is important for read performance. HBase has to decide which HFiles to choose for compaction. It decides this based on file size and number.

There are two types of Compactions:

- **Major Compaction:** In this type all HFiles of a column family in a given region are combined together to form a single HFile.

- **Minor Compaction:** Here, HBase takes only few HFiles and merges them into a single bigger HFile.

Compaction process requires heavy disk I/O which may lead to network congestion. This is called as Write Amplification. So, HBase does not schedule compaction process during peak hours (when the load on the system is heavy).

6.8 HBASE READ WRITE MECHANISM

HBase Write Mechanism:

Following are the steps in the process of a write operation in HBase:

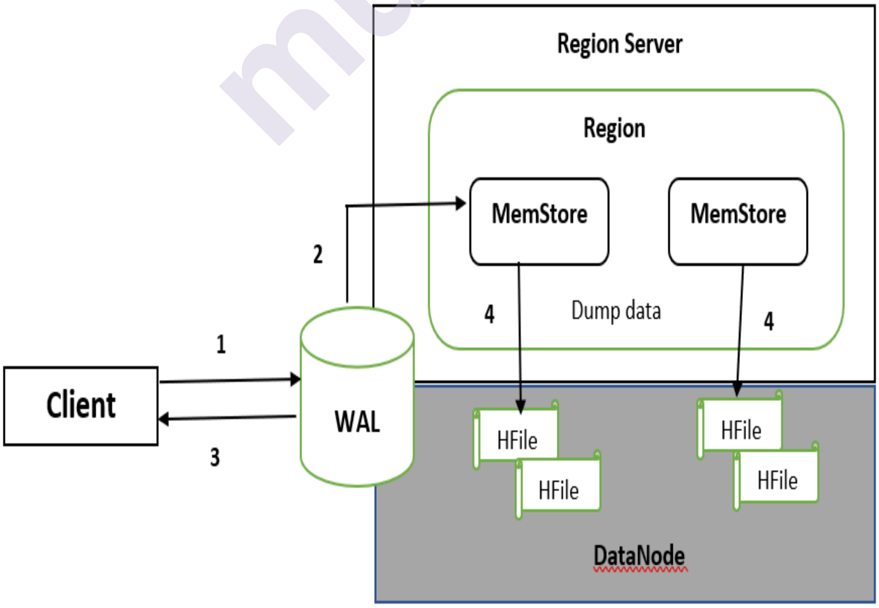
1. Client writes the data to Write Ahead Log (WAL) when it makes a write request to HBase. This new data is appended to the WAL file which is maintained on each Region Server.

2. After writing data to the WAL, data is then copied to MemStore. There is one MemStore for each column family. HBase writes data in WAL as well as MemStore to maintain durability of the data. The write is considered to be complete only after data is written to both places WAL and MemStore.

3. Client receives an acknowledgement once the data is stored in MemStore. The MemStore updates the data stored in it in lexicographical order as sorted Key values.

4. When Memstore fills up, it dumps the data to HFile in a sorted fashion. It does not write data to an existing HFile but instead creates a new HFile for every dump. HFiles are stored in HDFS (Hadoop Distributed File System). HBase contains multiple HFiles for each column family but a single HFile will never have data from multiple column families. In case of cluster crash, data which is not dumped to HFile from MemStore, is easily recovered from WAL as a part of HBase recovery process.

The below given diagram depicts the process of Write mechanism in HBase.



HBase Read Mechanism:

Read process commences as a client sends a read request to HBase.

HBase uses a LRU cache called as Block Cache. Each column family has its own Block cache. The Block cache stores frequently accessed data from HFiles so as to minimize disk reads. So, whenever a read request is sent by the client, the Block cache is first checked for the relevant row. If it is not found in Block cache then the HFiles on the disk are checked for data. Once the data is found in HFiles then it is copied to Block cache as client may need it again in near future. Finally, the required data is sent to client along with an acknowledgement.

6.9 EXERCISES

1. How NoSQL is used to deal with Bigdata?
2. Explain how Replication is used as one of the ways to deal with Bigdata in NoSQL?
3. Explain how Shared memory architecture differs from Shared Disk architecture?
4. Explain in detail Shared nothing architecture.
5. What are the advantages of Shared Nothing Architecture over other forms?
6. What are the two Distribution Models? Explain each one in short.
7. Compare Master-Slave distribution model to Peer-to-Peer model
8. What are the striking features of HBase which makes it different from its competitors?
9. What are the benefits of HBase over other NoSQL databases?
10. Explain how HBase is used in capturing incremental data?
11. Explain how social media platforms like Facebook, Meetup and Twitter use HBase as a part of their core infrastructure?
12. What are the major components of HBase Data model? Explain each one in brief.
13. What important role do Region Server and Zookeeper play in HBase architecture?
14. Explain Compaction along with its types.
15. How HBase handles client's write request?
16. Explain read mechanism in HBase.

6.10 ADDITIONAL REFERENCES

1. ‘Big Data for Dummies’ by Judith Hurwitz, Alan Nugent, Fern Halper, Marcia Kaufman – Wiley India

2. ‘Fundamentals of Business Analytics’ by RN Prasad, Seema Acharya – Wiley India

3. ‘HBase Essentials’ by Nishant Garg – PACKT Publishing

4. ‘HBase The Definitive Guide’ by Lars George – O’Reilly Publishers

5. www.ijert.org/handling-big-data-using-nosql-database

6. pld.cs.luc.edu/database/big_data.html

7. www.tutorialspoint.com/hbase/hbase_architecture.htm

8. www.upgrad.com/blog/hbase-architecture

9. https://towardsdatascience.com/hbase-working-principle-a-part-of-hadoop-architecture

*****

MODULE IV

7

HADOOP ECOSYSTEM: HIVE AND PIG

Unit Structure

7.0 OBJECTIVES

- Complete knowledge of Apache Hive.
- Able to learn work the Hive project independently.
- Understand Hive Architecture.
- Learn Hive to explore and analyse huge datasets.
- Learn HQL language.
- Able to learn different operations of Hive.

7.1 INTRODUCTION OF HIVE

The term 'Big Data' refers to massive datasets with a high volume, high velocity, and a diverse mix of data that is growing by the day. Processing

Big Data using typical data management solutions is tough. As a result, the Apache Software Foundation developed Hadoop, a framework for managing and processing large amounts of data. Hive is a Hadoop data warehouse infrastructure solution that allows you to process structured data. It is built on top of Hadoop to summarise Big Data and facilitate querying and analysis. Initially developed by Facebook, Hive was eventually taken up by the Apache Software Foundation and developed as an open-source project under the name Apache Hive. It is utilised by a variety of businesses.

7.2 ARCHITECTURE

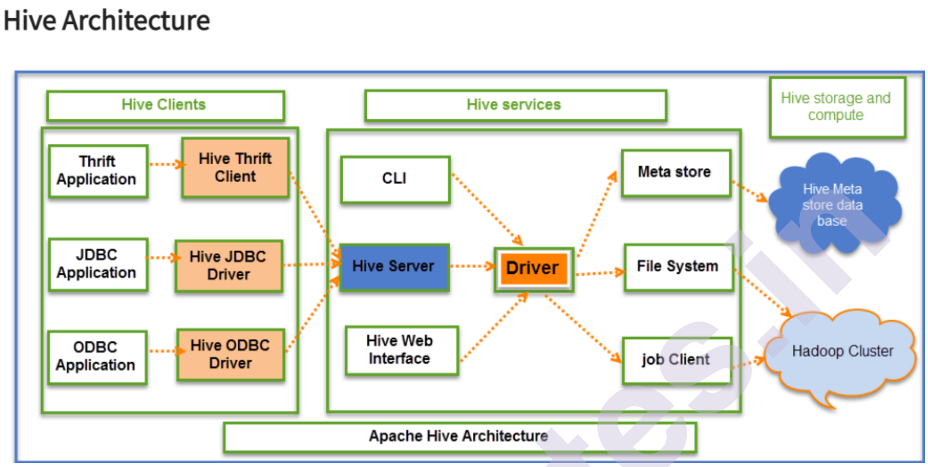


Figure 1: Architecture of HIVE

Important components of HIVE:

- HIVE clients
- HIVE Services
- Storage and computing

HIVE Clients:

Hive offers a variety of drivers for interacting with various types of applications. It will provide a Thrift client for communication in Thrift-based applications. JDBC Drivers are available for Java-based applications. ODBC drivers are available for any sort of application. In the Hive services, these Clients and Drivers communicate with the Hive server.

HIVE Services:

Hive Services can be used by clients to interact with Hive. If a client wishes to do any query-related actions in Hive, it must use Hive Services to do so. The command line interface (CLI) is used by Hive to perform DDL (Data Definition Language) operations. As indicated in the architectural diagram above, all drivers communicate with Hive server and the main driver in Hive services. The main driver is present in Hive services, and it interfaces with all types of JDBC, ODBC, and other client-specific applications. Driver will process requests from various apps and send them to the meta store and field systems for processing.

Storage and Computing:

Hive services like Meta store, File system, and Job Client communicate with Hive storage and carry out the following tasks.

- The Hive "Meta storage database" stores the metadata of tables generated in Hive.
- Query results and data loaded into tables will be stored on HDFS in a Hadoop cluster.

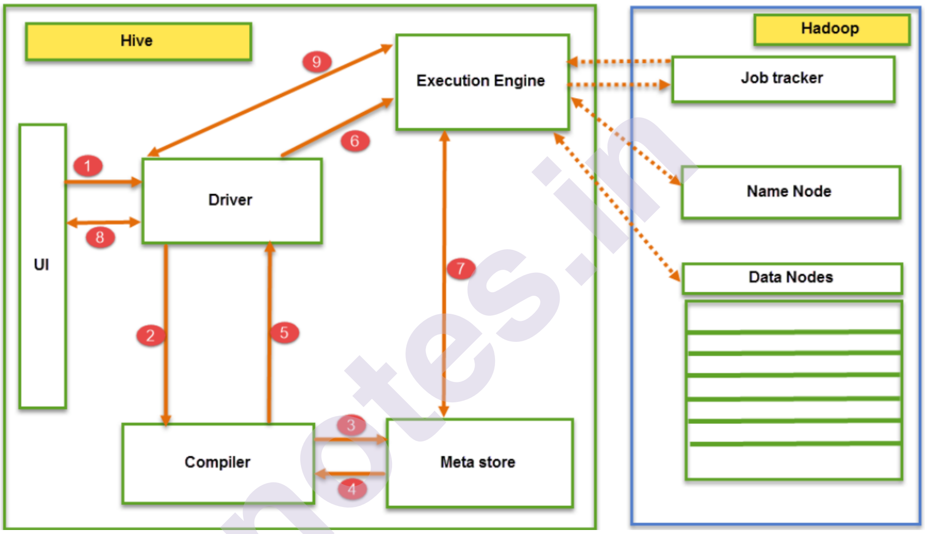


Figure 2: Job Execution Flow

From the above screenshot we can understand the Job execution flow in Hive with Hadoop the data flow in Hive behaves in the following pattern;

1. Executing Query from the UI (User Interface)
2. The driver is interacting with Compiler for getting the plan. (Here plan refers to query execution) process and its related metadata information gathering
3. The compiler creates the plan for a job to be executed. Compiler communicating with Meta store for getting metadata request
4. Meta store sends metadata information back to compiler
5. Compiler communicating with Driver with the proposed plan to execute the query
6. Driver Sending execution plans to Execution engine
7. Execution Engine (EE) acts as a bridge between Hive and Hadoop to process the query. For DFS operations.

- EE should first contacts Name Node and then to Data nodes to get the values stored in tables.

- EE is going to fetch desired records from Data Nodes. The actual data of tables resides in data node only. While from Name Node it only fetches the metadata information for the query.

- It collects actual data from data nodes related to mentioned query

- Execution Engine (EE) communicates bi-directionally with Meta store present in Hive to perform DDL (Data Definition Language) operations. Here DDL operations like CREATE, DROP and ALTERING tables and databases are done. Meta store will store information about database name, table names and column names only. It will fetch data related to query mentioned.

- Execution Engine (EE) in turn communicates with Hadoop daemons such as Name node, Data nodes, and job tracker to execute the query on top of Hadoop file system

8. Fetching results from driver

9. Sending results to Execution engine.

Once the results fetched from data nodes to the EE, it will send results back to driver and to UI (front end). Hive Continuously in contact with Hadoop file system and its daemons via Execution engine. The dotted arrow in the Job flow diagram shows the Execution engine communication with Hadoop daemons.

7.3 WAREHOUSE DIRECTORY AND META-DATA

When you create a table in Hive, by default Hive will manage the data, which means that Hive moves the data into its warehouse directory. Alternatively, you may create an external table, which tells Hive to refer to the data that is at an existing location outside the warehouse directory. The Hive component of the Hadoop eco-system processes all of the structure information of the many tables and partitions in the warehouse, including column, column type information, and the accompanying HDFS files where data is stored. That's where all of Hive's metadata is stored. Metadata which meta-store stores contain things like:

1. IDs of database

2. IDs of Tables and index

3. Time of creation of the index

4. Time of creation of tables

5. Input format used for tables

6. Output format used for tables

Apache Hive metadata is stored in Meta-store, which is a central repository. It uses a relational database to hold metadata for Hive tables (such as schema and location) and partitions. It uses the meta-store service API to give clients access to this data.

7.4 HIVE QUERY LANGUAG

The Hive Query Language (HiveQL) is a query language for processing and analysing structured data in Apache Hive. It shields users from Map Reduce programming's complexities. To make learning easier, it borrows notions from relational databases, such as tables, rows, columns, and schema. Hive provides a CLI for Hive query writing using Hive Query Language (HiveQL). HiveQL is Hive's query language, a dialect of SQL. It is heavily influenced by MySQL, so if you are familiar with MySQL, you should feel at home using Hive. Hive's SQL dialect, called HiveQL, is a mixture of SQL-92, MySQL, and Oracle's SQL dialect. The level of SQL-92 support has improved over time, and will likely continue to get better. HiveQL also provides features from later SQL standards, such as window functions (also known as analytic functions) from SQL:2003. Some of Hive's non-standard extensions to SQL were inspired by MapReduce, such as multitable inserts (see Multitable insert) and the TRANSFORM, MAP, and REDUCE clauses. When starting Hive for the first time, we can check that it is working by listing its tables — there should be none. The command must be terminated with a semicolon to tell Hive to execute it:

```
hive> SHOW TABLES;

OK Time taken: 0.473 seconds
```

Like SQL, HiveQL is generally case insensitive (except for string comparisons), so show tables; works equally well here. The Tab key will autocomplete Hive keywords and functions. For a fresh install, the command takes a few seconds to run as it lazily creates the meta-store database on your machine.

7.4.1 Loading Data Into Hive:

We can load data into hive table in three ways. Two of them are DML operations of Hive. Third way is using HDFS command. If we have data in RDBMS system like Oracle, Mysql, DB2 or SQL Server we can import it using SQOOP tool. Now to implement loading of data into hive we will practise some commands. Create a table called employee which has data as following:

Employee (eno, ename, salary, dno)

```
11, Balaji,100200,15

12, Radhika,120000,25

13, Nityanand,150000,15
```

14, Sai Nirupam,120000,35

Using Insert command:

We can load data into a table using Insert command in two ways. One Using Values command and other is using queries. Using Values command, we can append more rows of data into existing table. For example, to existing above employee table we can add extra row 15, Bala,150000,35 like below:

Insert into table employee values (15,'Bala',150000,35)

After this You can run a select command to see newly added row.

By using Queries:

You can also save the results of a query in a table. As an example, assuming you have an emp table, you can upload data into the employee database as seen below.

Insert into table employee Select * from emp where dno=45;

By using Load:

You can load data into a hive table using Load statement in two ways. One is from local file system to hive table and other is from HDFS to Hive table.

By using HDFS:

If you have data in a local file, you can easily upload it with HDFS commands. To find the location of a table, use the describe command, as shown below.

describe formatted employee;

It will display Location of the table, Assume You got location as /data/employee, you can upload data into table by using one of below commands.

hadoop fs -put /path/to/localfile /Data/employee

hadoop fs -copyFromLocal /path/to/localfile /Data/employee

hadoop fs -moveFromLocal /path/to/localfile /Data/employee

7.4.2 HIVE BUILT-IN FUNCTIONS

Return Type	Signature	Description
BIGINT	round(double a)	It returns the rounded BIGINT value of the double.

BIGINT	floor(double a)	It returns the maximum BIGINT value that is equal or less than the double.
BIGINT	ceil(double a)	It returns the minimum BIGINT value that is equal or greater than the double.
Double	rand(), rand(int seed)	It returns a random number that changes from row to row.
String	concat(string A, string B,...)	It returns the string resulting from concatenating B after A.
String	substr(string A, int start)	It returns the substring of A starting from start position till the end of string A.
String	substr(string A, int start, int length)	It returns the substring of A starting from start position with the given length.
String	upper(string A)	It returns the string resulting from converting all characters of A to upper case.
string	ucase(string A)	Same as above.
string	lower(string A)	It returns the string resulting from converting all characters of B to lower case.
string	lcase(string A)	Same as above.
string	trim(string A)	It returns the string resulting from trimming spaces from both ends of A.
string	ltrim(string A)	It returns the string resulting from trimming spaces from the beginning (left hand side) of A.
string	rtrim(string A)	rtrim(string A) It returns the string resulting from trimming spaces from the end (right hand side) of A.
string	regexp_replace(string A, string B, string C)	It returns the string resulting from replacing all substrings in B that match the Java regular expression syntax with C.
Int	size(Map<K.V>)	It returns the number of elements in the map type.

Int	size(Array<T>)	It returns the number of elements in the array type.
value of <type>	cast(<expr> as <type>)	It converts the results of the expression expr to <type> e.g. cast('1' as BIGINT) converts the string '1' to it integral representation. A NULL is returned if the conversion does not succeed.
string	from_unixtime(int unixtime)	convert the number of seconds from Unix epoch (1970-01-01 00:00:00 UTC) to a string representing the timestamp of that moment in the current system time zone in the format of "1970-01-01 00:00:00"
string	to_date(string timestamp)	It returns the date part of a timestamp string: to_date("1970-01-01 00:00:00") = "1970-01-01"
Int	year(string date)	It returns the year part of a date or a timestamp string: year("1970-01-01 00:00:00") = 1970, year("1970-01-01") = 1970
Int	month(string date)	It returns the month part of a date or a timestamp string: month("1970-11-01 00:00:00") = 11, month("1970-11-01") = 11
Int	day(string date)	It returns the day part of a date or a timestamp string: day("1970-11-01 00:00:00") = 1, day("1970-11-01") = 1
string	get_json_object(string json_string, string path)	It extracts json object from a json string based on json path specified, and returns json string of the extracted json object. It returns NULL if the input json string is invalid.

Some Examples of HIVE Built in functions:

Round() function:

```
hive> SELECT round(2.6) from temp;
```

On successful execution of query, you get to see the following response:

```
3.0
```

Floor() function:

```
hive> SELECT floor(2.6) from temp;
```

On successful execution of the query, you get to see the following response:

```
2.0
```

Ceil() function:

```
hive> SELECT ceil(2.6) from temp;
```

On successful execution of the query, you get to see the following response:

```
3.0
```

Aggregate Functions:

The following built-in aggregate functions are supported by Hive. These functions are used in the same way as SQL aggregate functions.

Return Type	Signature	Description
BIGINT	count(*), count(expr),	count(*) - Returns the total number of retrieved rows.
DOUBLE	sum(col), sum(DISTINCT col)	It returns the sum of the elements in the group or the sum of the distinct values of the column in the group.
DOUBLE	avg(col), avg(DISTINCT col)	It returns the average of the elements in the group or the average of the distinct values of the column in the group.
DOUBLE	min(col)	It returns the minimum value of the column in the group.
DOUBLE	max(col)	It returns the maximum value of the column in the group.

7.4.3 Joins in Hive:

- In Joins, only Equality joins are allowed.
- However, in the same query more than two tables can be joined.
- Basically, to offer more control over ON Clause for which there is no match LEFT, RIGHT, FULL OUTER joins exist in order.
- Also, note that Hive Joins are not Commutative
- Whether they are LEFT or RIGHT joins in Hive, even then Joins are left-associative.

SYNTAX:

Following is a syntax of Hive Join – HiveQL Select Clause.

```
join_table:
 table_reference JOIN table_factor [join_condition]
  | table_reference {LEFT|RIGHT|FULL} [OUTER] JOIN table_reference
  join_condition
  | table_reference LEFT SEMI JOIN table_reference join_condition
  | table_reference CROSS JOIN table_reference [join_condition]
```

Types of Joins:

- Inner join in Hive
- Left Outer Join in Hive
- Right Outer Join in Hive
- Full Outer Join in Hive

Inner Join:

Basically, to combine and retrieve the records from multiple tables we use Hive Join clause. Moreover, in SQL JOIN is as same as OUTER JOIN. Moreover, by using the primary keys and foreign keys of the tables JOIN condition is to be raised. Furthermore, the below query executes JOIN the CUSTOMER and ORDER tables. Then further retrieves the records:

```
hive> SELECT c.ID, c.NAME, c.AGE, o.AMOUNT

FROM CUSTOMERS c JOIN ORDERS o

ON (c.ID = o.CUSTOMER_ID);
```

Left Outer Join:

On defining HiveQL Left Outer Join, even if there are no matches in the right table it returns all the rows from the left table. To be more specific, even if the ON clause matches 0 (zero) records in the right table, then also this Hive JOIN still returns a row in the result. Although, it returns with NULL in each column from the right table. In addition, it returns all the values from the left table. Also, the matched values from the right table, or NULL in case of no matching JOIN predicate. However, the below query shows LEFT OUTER JOIN between CUSTOMER as well as ORDER tables:

```
hive> SELECT c.ID, c.NAME, o.AMOUNT, o.DATE

FROM CUSTOMERS c

LEFT OUTER JOIN ORDERS o

ON (c.ID = o.CUSTOMER_ID);
```

Right Outer Join:

Basically, even if there are no matches in the left table, HiveQL Right Outer Join returns all the rows from the right table. To be more specific, even if the ON clause matches 0 (zero) records in the left table, then also this Hive JOIN still returns a row in the result. Although, it returns with NULL in each column from the left table. In addition, it returns all the values from the right table. Also, the matched values from the left table or NULL in case of no matching join predicate.

notranslate"> hive> SELECT c.ID, c.NAME, o.AMOUNT, o.DATE FROM CUSTOMERS c RIGHT OUTER JOIN ORDERS o ON (c.ID = o.CUSTOMER_ID);

Full Outer Join:

The primary goal of this HiveQL Full outer Join is to merge the records from both the left and right outside tables in order to satisfy the Hive JOIN requirement. In addition, this connected table either contains all of the records from both tables or fills in NULL values for any missing matches on either side.

```
hive> SELECT c.ID, c.NAME, o.AMOUNT, o.DATE

FROM CUSTOMERS c

FULL OUTER JOIN ORDERS o

ON (c.ID = o.CUSTOMER_ID);
```

7.4.4 Partitioning in Hive:

Table partitioning is the process of splitting table data into sections based on the values of specific columns, such as date or country, and then separating the input records into distinct files/directories based on the date or country.

Partitioning can be done on the basis of many columns, resulting in a multi-dimensional structure on directory storage. For example, in addition to separating log entries by date column, we can also partition single-day records into country-specific separate files by inserting the country column in the partitioning. In the examples, we'll see more about this. Partitions are defined using the PARTITIONED BY clause and a set of column definitions for partitioning at the time of table formation. Partitions are defined using the PARTITIONED BY clause and a set of column definitions for partitioning at the time of table formation.

Advantages:

- Partitioning is used for distributing execution load horizontally.
- As the data is stored as slices/parts, query response time is faster to process the small part of the data instead of looking for a search in the entire data set.

- For example, in a large user table where the table is partitioned by country, then selecting users of country 'IN' will just scan one directory 'country=IN' instead of all the directories.

Limitations:

Having too many partitions in table creates large number of files and directories in HDFS, which is an overhead to NameNode since it must keep all metadata for the file system in memory only. Partitions may optimize some queries based on WHERE clauses, but may be less responsive for other important queries on grouping clauses. In Map reduce processing, huge number of partitions will lead to huge no of tasks (which will run in separate JVM) in each map reduce job, thus creates lot of overhead in maintaining JVM start up and tear down. For small files, a separate task will be used for each file. In worst scenarios, the overhead of JVM start up and tear down can exceed the actual processing time.

7.4.5 Querying Data:

Sorting and Aggregating:

Sorting data in Hive can be achieved by using a standard ORDER BY clause. ORDER BY performs a parallel total sort of the input (like that described in Total Sort). When a globally sorted result is not required — and in many cases it isn't you can use Hive's nonstandard extension, SORT BY, instead. SORT BY produces a sorted file per reducer. In some cases, you want to control which reducer a particular row goes to — typically so you can perform some subsequent aggregation. This is what Hive's DISTRIBUTE BY clause does. Here's an example to sort the weather dataset by year and temperature, in such a way as to ensure that all the rows for a given year end up in the same reducer partition:

```
hive> FROM records2

 > SELECT year, temperature

> DISTRIBUTE BY year

 > SORT BY year ASC, temperature DESC;

1949 111

1949 78

1950 22

1950 0

1950 -11
```

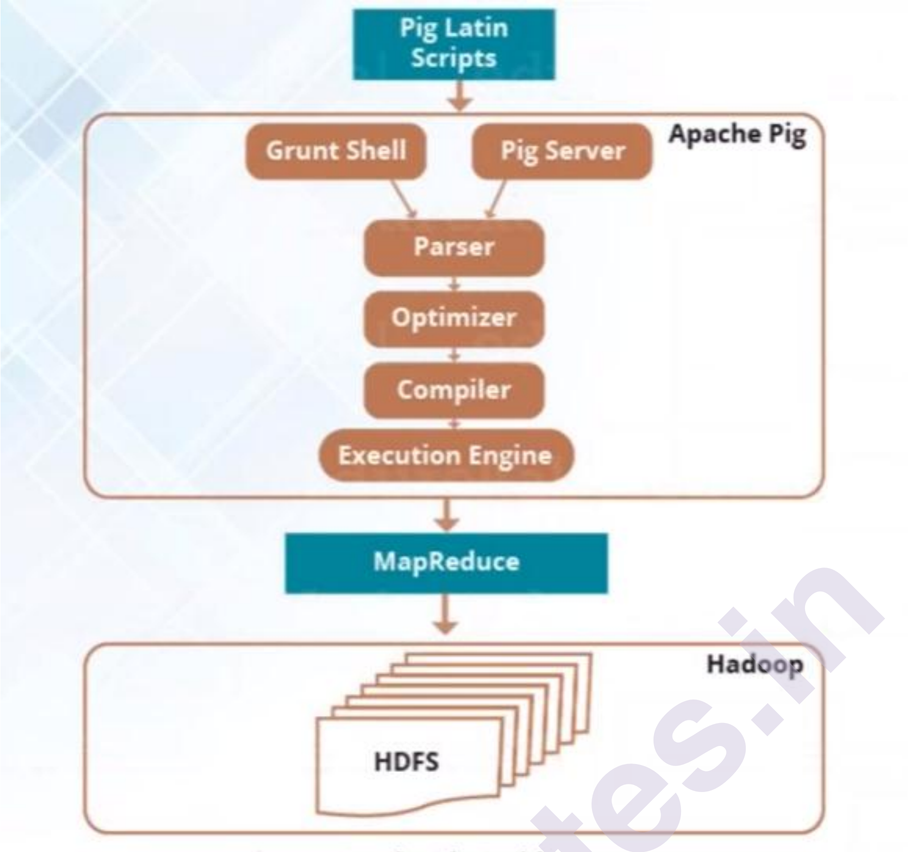
A follow-on query (or a query that nests this query as a subquery; see Subqueries) would be able to use the fact that each year's temperatures were grouped and sorted (in descending order) in the same file. If the columns for SORT BY and DISTRIBUTE BY are the same, you can use CLUSTER BY as a shorthand for specifying both.

7.5 PIG

7.5.1 Background:

Apache Pig raises the level of abstraction for processing large datasets. MapReduce allows you, as the programmer, to specify a map function followed by a reduce function, but working out how to fit your data processing into this pattern, which often requires multiple MapReduce stages, can be a challenge. With Pig, the data structures are much richer, typically being multivalued and nested, and the transformations you can apply to the data are much more powerful. They include joins, for example, which are not for the faint of heart in MapReduce. Pig is made up of two pieces: The language used to express data flows, called Pig Latin. The execution environment to run Pig Latin programs. There are currently two environments: local execution in a single JVM and distributed execution on a Hadoop cluster. A Pig Latin program is made up of a series of operations, or transformations, that are applied to the input data to produce output. Taken as a whole, the operations describe a data flow, which the Pig execution environment translates into an executable representation and then runs. Under the covers, Pig turns the transformations into a series of MapReduce jobs, but as a programmer you are mostly unaware of this, which allows you to focus on the data rather than the nature of the execution. Pig is a scripting language for exploring large datasets. One criticism of MapReduce is that the development cycle is very long. Writing the mappers and reducers, compiling and packaging the code, submitting the job(s), and retrieving the results is a time-consuming business, and even with Streaming, which removes the compile and package step, the experience is still involved. Pig's sweet spot is its ability to process terabytes of data in response to a half-dozen lines of Pig Latin issued from the console. Indeed, it was created at Yahoo! to make it easier for researchers and engineers to mine the huge datasets there. Pig is very supportive of a programmer writing a query, since it provides several commands for introspecting the data structures in your program as it is written. Even more useful, it can perform a sample run on a representative subset of your input data, so you can see whether there are errors in the processing before unleashing it on the full dataset. Pig was designed to be extensible. Virtually all parts of the processing path are customizable: loading, storing, filtering, grouping, and joining can all be altered by userdefined functions (UDFs). These functions operate on Pig's nested data model, so they can integrate very deeply with Pig's operators. As another benefit, UDFs tend to be more reusable than the libraries developed for writing MapReduce programs.

7.5.2 Pig Architecture:



Apache Pig Architecture

1. **Parser:** The parser handles any pig scripts or instructions in the grunt shell. Parse will run checks on the scripts, such as checking the syntax, type checking, and a variety of other things. These checks will produce results in the form of a Directed Acyclic Graph (DAG), which contains pig Latin sentences and logical operators. Our logical operators of the scripts are nodes, and data flows are edges, therefore the DAG will have nodes that are connected to distinct edges.

2. **Optimizer:** After parsing and DAG generation, the DAG is sent to the logical optimizer, which performs logical optimizations such as projection and pushdown. By removing extraneous columns or data and pruning the loader to only load the relevant column, projection and pushdown increase query performance.

3. **Compiler:** The compiler compiles the optimised logical plan provided above and generates a series of Map-Reduce tasks. Essentially, the compiler will transform pig jobs into MapReduce jobs and exploit optimization opportunities in scripts, allowing the programmer to avoid manually tuning the software. Pig's compiler can reorder the execution sequence to enhance performance if the execution plan remains the same as the original programme because it is a data-flow language.

4. **Execution Engine:** Finally, all of the MapReduce jobs generated by the compiler are sorted and delivered to Hadoop. Finally, Hadoop executes the MapReduce job to produce the desired output.

5. Pig has two execution modes, which are determined by the location of the script and the availability of data.

- **Local Mode:** For limited data sets, local mode is the best option. Pig is implemented on a single JVM because all files are installed and run-on localhost, preventing parallel mapper execution. Pig will also peek into the local file system while importing data.

- **MapReduce Mode (MR Mode):** In MapReduce, the mode programmer needs access and setup of the Hadoop cluster and HDFS installation. In this mode data on which processing is done is exists in the HDFS system. After execution of pig script in MR mode, pig Latin statement is converted into Map Reduce jobs in the back-end to perform the operations on the data

7.5.3 Latin Basics:

Pig Latin is the language used to analyse data in Hadoop using Apache Pig. In this chapter, we are going to discuss the basics of Pig Latin such as Pig Latin statements, data types, general and relational operators, and Pig Latin UDF's.

Pig Statements:

While processing data using Pig Latin, statements are the basic constructs.

- These statements work with relations. They include expressions and schemas.

- Every statement ends with a semicolon (;).

- We will perform various operations using operators provided by Pig Latin, through statements.

- Except LOAD and STORE, while performing all other operations, Pig Latin statements take a relation as input and produce another relation as output.

- As soon as you enter a Load statement in the Grunt shell, its semantic checking will be carried out. To see the contents of the schema, you need to use the Dump operator. Only after performing the dump operation, the MapReduce job for loading the data into the file system will be carried out.

7.5.4 Pig Execution Modes:

There are three execution modes for pig which are

- **Interactive Mode (Grunt shell):** You can run Apache Pig in interactive mode using the Grunt shell. In this shell, you can enter the Pig Latin statements and get the output (using Dump operator).
- **Batch Mode (Script):** You can run Apache Pig in Batch mode by writing the Pig Latin script in a single file with **.pig** extension.
- **Embedded Mode (UDF):** Apache Pig provides the provision of defining our own functions (**U**ser **D**efined **F**unctions) in programming languages such as Java, and using them in our script.

7.5.5 Pig Processing – Loading and Transforming Data:

Apache Pig, in general, runs on top of Hadoop. It's a statistical tool for analysing huge datasets in the Hadoop File System. To use Apache Pig to analyse data, we must first load the data into Apache Pig. This chapter explains how to load data from HDFS into Apache Pig.

LOAD operator:

The LOAD operator of Pig Latin can be used to load data into Apache Pig from a file system (HDFS/ Local). The "=" operator divides the load statement into two sections. On the left, we must specify the name of the relation in which we want to store the data, and on the right, we must specify how we will store the data. The Load operator's syntax is seen below.

Relation_name = LOAD 'Input file path' USING function as schema;

- **relation_name:** We have to mention the relation in which we want to store the data.
- **Input file path:** We have to mention the HDFS directory where the file is stored. (In MapReduce mode)
- **Function:** We have to choose a function from the set of load functions provided by Apache Pig (BinStorage, JsonLoader, PigStorage, TextLoader).
- **Schema:** We have to define the schema of the data. We can define the required schema as follows:

(column1: data type, column2 : data type, column3 : data type);

Store Operator:

You can store the loaded data in the file system using the store operator. This chapter explains how to store data in Apache Pig using the Store operator.

Syntax:

STORE Relation_name INTO ' required_directory_path ' [USING function];

Loading and Storing Data, we have seen how to load data from external storage for processing in Pig. Storing the results is straightforward, too. Here's an example of using PigStorage to store tuples as plain-text values separated by a colon character: grunt> STORE A INTO 'out' USING PigStorage(':');

```
grunt> cat out

Joe:cherry:2

Ali:apple:3

 Joe:banana:2

Eve:apple:7
```

7.5.6 Pig Built-In Functions:

Type	Examples
EVAL functions	AVG, COUNT, COUNT_STAR, SUM, TOKENIZE, MAX, MIN, SIZE etc
LOAD or STORE functions	Pigstorage(), Textloader, HbaseStorage, JsonLoader, JsonStorage etc
Math functions	ABS, COS, SIN, TAN, CEIL, FLOOR, ROUND, RANDOM etc
String functions	TRIM, RTRIM, SUBSTRING, LOWER, UPPER etc
DateTime function	GetDay, GetHour, GetYear, ToUnixTime, ToString etc

Eval Functions:

AVG(col): computes the average of the numerical values in a single column of a bag

CONCAT(string expression1, string expression2): Concatenates two expressions of identical type

COUNT(DataBag bag): Computes the number of elements in a bag excluding null values

COUNT STAR (DataBag bag1, DataBag bag 2): Computes the number of elements in a bag including null values.

DIFF(DataBag bag1, DataBag bag2): It is used to compare two bags, if any element in one bag is not present in the other bag are returned in a bag

IsEmpty(DataBag bag), IsEmpty(Map map): It is used to check if the bag or map is empty

Max(col): Computes the maximum of the numeric values or character in a single column bag

MIN(col): Computes the minimum of the numeric values or character in a single column bag

DEFINE pluck pluckTuple(expression1): It allows the user to specify a string prefix, and filters the columns which begins with that prefix

SIZE(expression): Computes the number of elements based on any pig data

SUBSTRACT(DataBag bag1, DataBag bag2): It returns the bag which does not contain bag1 element in bag2

SUM: Computes the sum of the values in a single-column bag

TOKENIZE(String expression[,'field delimiter'): It splits the string and outputs a bag of words.

Filtering:

The FILTER operator is used to filter tuples from a relation depending on a criterion.

Syntax:

```
grunt> Relation2_name = FILTER Relation1_name BY (condition);
```

For example:

StudentInfo.txt

```
1,Rajiv,Reddy,21,9848022337,Hyderabad

2,siddarth,Battacharya,22,9848022338,Kolkata

3,Rajesh,Khanna,22,9848022339,Delhi

4,Preethi,Agarwal,21,9848022330,Pune

5,Trupthi,Mohanthy,23,9848022336,Bhuwaneshwar

6,Archana,Mishra,23,9848022335,Chennai

7,Komal,Nayak,24,9848022334,trivendram

8,Bharathi,Nambiayar,24,9848022333,Chennai
```

Loading of file **StudentInfo.txt**

```
grunt> student_details = LOAD
'hdfs://localhost:9000/pig_data/StudentInfo.txt' USING PigStorage(',') as
```

(id:int, firstname:chararray, lastname:chararray, age:int, phone:chararray, city:chararray);

Let’s use the Filter operator to get the details of the students who belong to the city Chennai.

filter_data = FILTER student_details BY city == 'Chennai';

7.5.7 Grouping and Sorting:

To group data in one or more relations, use the GROUP operator. It gathers information using the same key.

grunt> Group_data = GROUP Relation_name BY age;

The ORDER BY operator is used to sort the contents of a relation according to one or more fields.

Syntax:

grunt> Relation_name2 = ORDER Relatin_name1 BY (ASC|DESC);

7.5.8 Installation of Pig and Pig Latin Commands:

Before you start using Apache Pig, you must have Hadoop and Java installed on your PC. As a result, before installing Apache Pig, install java and Hadoop.

First of all, download the latest version of Apache Pig from the following website − https://pig.apache.org/

Install Apache Pig:

After downloading the Apache Pig software, install it in your Linux environment by following the steps given below.

STEP 1:

Create a directory with the name Pig in the same directory where the installation directories of Hadoop, Java, and other software were installed. (In our tutorial, we have created the Pig directory in the user named Hadoop).

```
$ mkdir Pig
```

STEP 2:

Extract the downloaded tar files as shown below.

```
$ cd Downloads/

$ tar zxvf pig-0.15.0-src.tar.gz

$ tar zxvf pig-0.15.0.tar.gz
```

STEP 3:

Move the content of pig-0.15.0-src.tar.gz file to the Pig directory created earlier as shown below.

```
$ mv pig-0.15.0-src.tar.gz/* /home/Hadoop/Pig/
```

Configure Apache Pig:

After installing Apache Pig, we have to configure it. To configure, we need to edit two files − bashrc and pig.properties.

.bashrc file

In the .bashrc file, set the following variables:

- PIG_HOME folder to the Apache Pig's installation folder,
- PATH environment variable to the bin folder, and
- PIG_CLASSPATH environment variable to the etc (configuration) folder of your Hadoop installations (the directory that contains the core-site.xml, hdfs-site.xml and mapred-site.xml files).

```
export PIG_HOME = /home/Hadoop/Pig

export PATH = $PATH:/home/Hadoop/pig/bin

export PIG_CLASSPATH = $HADOOP_HOME/conf
```

pig.properties file

In the conf folder of Pig, we have a file named pig.properties. In the pig.properties file, you can set various parameters as given below.

```
pig -h properties
```

7.6 LIST OF REFERENCES

- Tom White, “HADOOP: The definitive Guide” O Reilly 2012, Third Edition,ISBN: 978-1-449-31152-0
- Chuck Lam, “Hadoop in Action”, Dreamtech Press 2016, First Edition ,ISBN:139788177228137
- Shiva Achari,” Hadoop Essential “ PACKT Publications, ISBN 978-1-78439-668-8
- RadhaShankarmani and M. Vijayalakshmi ,”Big Data Analytics “Wiley Textbook Series, Second Edition, ISBN 9788126565757

Web References:

- https://hadoop.apache.org/docs/stable/
- https://pig.apache.org/

- https://hive.apache.org/
- https://www.guru99.com/introduction-hive.html

7.7 UNIT EXERCISES

- What is HIVE? Explain its architecture.
- Write a short note on warehouse directory and meta store.
- What is HIVE query language? Explain Built in functions in HIVE.
- How data is sorted and aggregated in HIVEQL?
- What is PIG? Explain its architecture in detail.

*****

8

KAFKA FUNDAMENTALS, KAFKA ARCHITECTURE

Unit Structure

8.0 OBJECTIVES

- To build real-time streaming data pipelines and real-time streaming applications
- To ingest and store streaming data while serving reads for the applications powering the data pipeline

- To get insight of components of Kafka.
- To describe cluster architecture
- To explore benefits of Kafka.

8.1 Introduction

In Big Data, an enormous volume of data is used. Regarding data, we have two main challenges. The first challenge is how to collect large volume of data and the second challenge is to analyse the collected data. To overcome those challenges, you must need a messaging system.

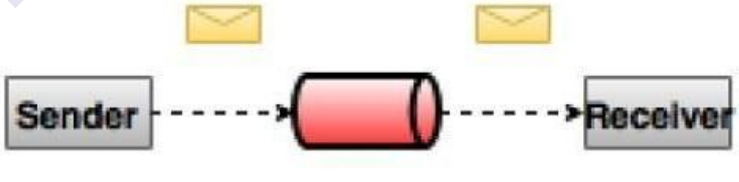
Kafka is designed for distributed high throughput systems. Kafka tends to work very well as a replacement for a more traditional message broker. In comparison to other messaging systems, Kafka has better throughput, built-in partitioning, replication, and inherent fault-tolerance, which makes it a good fit for large-scale message processing applications.

8.2 Messaging System

A Messaging System is responsible for transferring data from one application to another, so the applications can focus on data, but not worry about how to share it. Distributed messaging is based on the concept of reliable message queuing. Messages are queued asynchronously between client applications and messaging system. Two types of messaging patterns are available − one is point to point and the other is publish-subscribe (pub-sub) messaging system. Most of the messaging patterns follow pub-sub.

8.3 Point to Point Messaging System

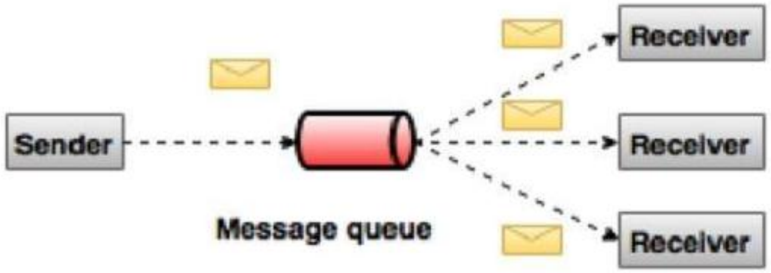
In a point-to-point system, messages are persisted in a queue. One or more consumers can consume the messages in the queue, but a particular message can be consumed by a maximum of one consumer only.



Message queue

Once a consumer reads a message in the queue, it disappears from that queue. The typical example of this system is an Order Processing System, where each order will be processed by one Order Processor, but Multiple Order Processors can work as well at the same time. The following diagram depicts the structure.

8.4 Publish-Subscribe Messaging System



In the publish-subscribe system, messages are persisted in a topic. Unlike point-to-point system, consumers can subscribe to one or more topic and consume all the messages in that topic. In the Publish-Subscribe system, message producers are called publishers and message consumers are called subscribers. A real-life example is Dish TV, which publishes different channels like sports, movies, music, etc., and anyone can subscribe to their own set of channels and get them whenever their subscribed channels are available.

8.5 What is Kafka?

Apache Kafka is a distributed publish-subscribe messaging system and a robust queue that can handle a high volume of data and enables you to pass messages from one endpoint to another. Kafka is suitable for both offline and online message consumption. Kafka messages are persisted on the disk and replicated within the cluster to prevent data loss. Kafka is built on top of the ZooKeeper synchronization service. It integrates very well with Apache Storm and Spark for real-time streaming data analysis.

8.6 Benefits

Following are a few benefits of Kafka:

- **Reliability:** Kafka is distributed, partitioned, replicated and fault tolerance.
- **Scalability:** Kafka messaging system scales easily without down time.
- **Durability:** Kafka uses Distributed commit log which means messages persists on disk as fast as possible, hence it is durable.
- **Performance:** Kafka has high throughput for both publishing and subscribing messages. It maintains stable performance even many TB of messages are stored.

Kafka is very fast and guarantees zero downtime and zero data loss.

8.7 USE CASES

Kafka can be used in many Use Cases. Some of them are listed below:

- **Metrics:** Kafka is often used for operational monitoring data. This involves aggregating statistics from distributed applications to produce centralized feeds of operational data.

- **Log Aggregation Solution:** Kafka can be used across an organization to collect logs from multiple services and make them available in a standard format to multiple consumers.

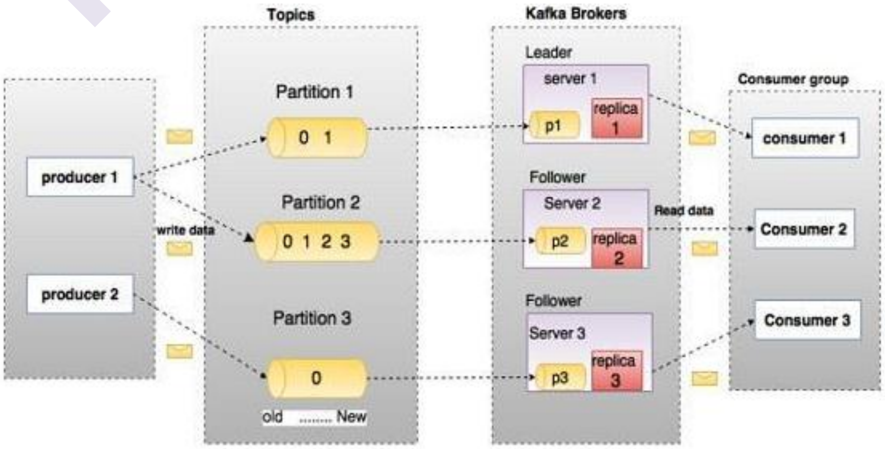
- **Stream Processing:** Popular frameworks such as Storm and Spark Streaming read data from a topic, processes it, and write processed data to a new topic where it becomes available for users and applications. Kafka's strong durability is also very useful in the context of stream processing.

8.8 NEED FOR KAFKA

Kafka is a unified platform for handling all the real-time data feeds. Kafka supports low latency message delivery and gives guarantee for fault tolerance in the presence of machine failures. It has the ability to handle a large number of diverse consumers. Kafka is very fast, performs 2 million writes/sec. Kafka persists all data to the disk, which essentially means that all the writes go to the page cache of the OS (RAM). This makes it very efficient to transfer data from page cache to a network socket.

8.9 KAFKA FUNDAMENTALS

Before moving deep into the Kafka, you must be aware of the main terminologies such as topics, brokers, producers, and consumers. The following diagram illustrates the main terminologies, and the table describes the diagram components in detail.



In the above diagram, a topic is configured into three partitions. Partition 1 has two offset factors 0 and 1. Partition 2 has four offset factors 0, 1, 2, and 3. Partition 3 has one offset factor 0. The id of the replica is same as the id of the server that hosts it.

Assume, if the replication factor of the topic is set to 3, then Kafka will create 3 identical replicas of each partition and place them in the cluster to make available for all its operations. To balance a load in cluster, each broker stores one or more of those partitions. Multiple producers and consumers can publish and retrieve messages at the same time.

Components and Description:

8.9.1 Topics:

A stream of messages belonging to a particular category is called a topic. Data is stored in topics.

Topics are split into partitions. For each topic, Kafka keeps a mini mum of one partition. Each such partition contains messages in an immutable ordered sequence. A partition is implemented as a set of segment files of equal sizes.

8.9.2 Partition:

Topics may have many partitions, so it can handle an arbitrary amount of data.

8.9.3 Partition offset:

Each partitioned message has a unique sequence id called as offset.

8.9.4 Replicas of partition:

Replicas are nothing but backups of a partition. Replicas are never read or write data. They are used to prevent data loss

8.9.5 Brokers:

- Brokers are simple system responsible for maintaining the published data. Each broker may have zero or more partitions per topic. Assume, if there are N partitions in a topic and N number of brokers, each broker will have one partition.
- Assume if there are N partitions in a topic and more than N brokers (n + m), the first N broker will have one partition and the next M broker will not have any partition for that particular topic.
- Assume if there are N partitions in a topic and less than N brokers (n-m), each broker will have one or more partition sharing among them. This scenario is not recommended due to unequal load distribution among the brokers.

8.9.6 Kafka Cluster:

Kafka's having more than one broker are called as Kafka cluster. A Kafka cluster can be expanded without downtime. These clusters are used to manage the persistence and replication of message data.

8.9.7 Producers:

Producers are the publisher of messages to one or more Kafka topics. Producers send data to Kafka brokers. Every time a producer pub-lishes a message to a broker, the broker simply appends the message to the last segment file. Actually, the message will be appended to a partition. Producer can also send messages to a partition of their choice.

8.9.8 Consumers:

Consumers read data from brokers. Consumers subscribes to one or more topics and consume published messages by pulling data from the brokers.

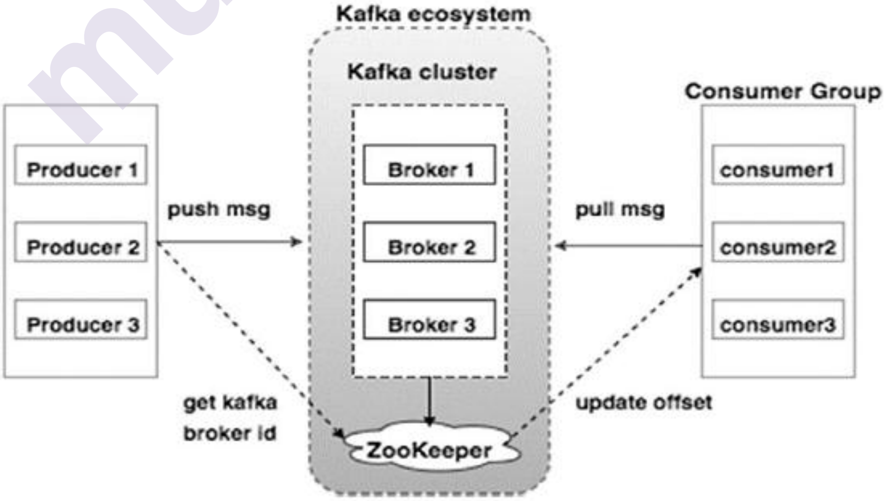
8.9.9 Leader:

Leader is the node responsible for all reads and writes for the given partition. Every partition has one server acting as a leader.

8.9.10 Follower:

Node which follows leader instructions are called as follower. If the leader fails, one of the followers will automatically become the new leader. A follower acts as normal consumer, pulls messages and up-dates its own data store.

8.10 CLUSTER ARCHITECTURE



Components and Description:

Broker:

Kafka cluster typically consists of multiple brokers to maintain load balance. Kafka brokers are stateless, so they use ZooKeeper for maintaining their cluster state. One Kafka broker instance can handle hundreds of thousands of reads and writes per second and each bro-ker can handle TB of messages without performance impact. Kafka broker leader election can be done by ZooKeeper.

ZooKeeper:

ZooKeeper is used for managing and coordinating Kafka broker. ZooKeeper service is mainly used to notify producer and consumer about the presence of any new broker in the Kafka system or failure of the broker in the Kafka system. As per the notification received by the Zookeeper regarding presence or failure of the broker then pro-ducer and consumer takes decision and starts coordinating their task with some other broker.

Producers:

Producers push data to brokers. When the new broker is started, all the producers search it and automatically sends a message to that new broker. Kafka producer doesn't wait for acknowledgements from the broker and sends messages as fast as the broker can handle.

Consumers:

Since Kafka brokers are stateless, which means that the consumer has to maintain how many messages have been consumed by using partition offset. If the consumer acknowledges a particular message offset, it implies that the consumer has consumed all prior messages. The consumer issues an asynchronous pull request to the broker to have a buffer of bytes ready to consume. The consumers can rewind or skip to any point in a partition simply by supplying an offset value. Consumer offset value is notified by ZooKeeper.

As of now, we discussed the core concepts of Kafka. Let us now throw some light on the workflow of Kafka.

Kafka is simply a collection of topics split into one or more partitions. A Kafka partition is a linearly ordered sequence of messages, where each message is identified by their index (called as offset). All the data in a Kafka cluster is the disjointed union of partitions. Incoming messages are written at the end of a partition and messages are sequentially read by consumers. Durability is provided by replicating messages to different brokers.

Kafka provides both pub-sub and queue-based messaging system in a fast, reliable, persisted, fault-tolerance and zero downtime manner. In both cases, producers simply send the message to a topic and consumer can choose any one type of messaging system depending on their need. Let us follow the steps in the next section to understand how the consumer can choose the messaging system of their choice.

8.11 Workflow of Pub-Sub Messaging

Following is the step wise workflow of the Pub-Sub Messaging −

- Producers send message to a topic at regular intervals.
- Kafka broker stores all messages in the partitions configured for that particular topic. It ensures the messages are equally shared between partitions. If the producer sends two messages and there are two partitions, Kafka will store one message in the first partition and the second message in the second partition.
- Consumer subscribes to a specific topic.
- Once the consumer subscribes to a topic, Kafka will provide the current offset of the topic to the consumer and also saves the offset in the Zookeeper ensemble.
- Consumer will request the Kafka in a regular interval (like 100 Ms) for new messages.
- Once Kafka receives the messages from producers, it forwards these messages to the consumers.
- Consumer will receive the message and process it.
- Once the messages are processed, consumer will send an acknowledgement to the Kafka broker.
- Once Kafka receives an acknowledgement, it changes the offset to the new value and updates it in the Zookeeper. Since offsets are maintained in the Zookeeper, the consumer can read next message correctly even during server outrages.
- This above flow will repeat until the consumer stops the request.
- Consumer has the option to rewind/skip to the desired offset of a topic at any time and read all the subsequent messages.

8.12 WORKFLOW OF QUEUE MESSAGING / CONSUMER GROUP

In a queue messaging system instead of a single consumer, a group of consumers having the same Group ID will subscribe to a topic. In simple terms, consumers subscribing to a topic with same Group ID are considered as a single group and the messages are shared among them. Let us check the actual workflow of this system.

- Producers send message to a topic in a regular interval.

- Kafka stores all messages in the partitions configured for that particular topic similar to the earlier scenario.

- A single consumer subscribes to a specific topic, assume Topic-01 with Group ID as Group-1.

- Kafka interacts with the consumer in the same way as Pub-Sub Messaging until new consumer subscribes the same topic, Topic-01 with the same Group ID as Group-1.

- Once the new consumer arrives, Kafka switches its operation to share mode and shares the data between the two consumers. This sharing will go on until the number of consumers reach the number of partitions configured for that particular topic.

- Once the number of consumers exceeds the number of partitions, the new consumer will not receive any further message until any one of the existing consumers unsubscribes. This scenario arises because each consumer in Kafka will be assigned a minimum of one partition and once all the partitions are assigned to the existing consumers, the new consumers will have to wait.

This feature is also called as Consumer Group. In the same way, Kafka will provide the best of both the systems in a very simple and efficient manner.

8.13 ROLE OF ZOOKEEPER

A critical dependency of Apache Kafka is Apache Zookeeper, which is a distributed configuration and synchronization service. Zookeeper serves as the coordination interface between the Kafka brokers and consumers. The Kafka servers share information via a Zookeeper cluster. Kafka stores basic metadata in Zookeeper such as information about topics, brokers, consumer offsets (queue readers) and so on.

Since all the critical information is stored in the Zookeeper and it normally replicates this data across its ensemble, failure of Kafka broker / Zookeeper does not affect the state of the Kafka cluster. Kafka will restore the state once the Zookeeper restarts. This gives zero downtime for Kafka. The leader election between the Kafka broker is also done by using Zookeeper in the event of leader failure.

8.14 LET US SUM UP

- Kafka is designed for distributed high throughput systems. Kafka tends to work very well as a replacement for a more traditional message broker.

- Apache Kafka is a distributed publish-subscribe messaging system and a robust queue that can handle a high volume of data and enables you to pass messages from one endpoint to another.

- Kafka supports low latency message delivery and gives guarantee for fault tolerance in the presence of machine failures.
- The main terminologies such as topics, brokers, producers, and consumers.
- Kafka cluster typically consists of multiple brokers to maintain load balance. Kafka brokers are stateless, so they use ZooKeeper for maintaining their cluster state.
- ZooKeeper service is mainly used to notify producer and consumer about the presence of any new broker in the Kafka system or failure of the broker in the Kafka system.
- In overall Kafka provides both pub-sub and queue-based messaging system in a fast, reliable, persisted, fault-tolerance and zero downtime manner.

8.15 LIST OF REFERENCES

- Kafka: The Definitive Guide: Real-Time Data and Stream Processing at Scale, book by Neha Narkhede
- Mastering Kafka Streams and KsqlDB,Book by Mitch Seymour
- Kafka,Book by Gwen Shapira, Neha Narkhede, and Todd Palino

8.16 BIBLIOGRAPHY

- https://www.tutorialspoint.com/apache_kafka/apache_kafka_fundamentals.htm
- https://medium.com/inspiredbrilliance/kafka-basics-and-core-concepts-5fd7a68c3193
- https://developer.ibm.com/articles/event-streams-kafka-fundamentals/
- Kafka: The Definitive Guide: Real-Time Data and Stream Processing at Scale, book by Neha Narkhede
- Kafka,Book by Gwen Shapira, Neha Narkhede, and Todd Palino

8.17 UNIT END EXERCISE

1. What is Apache Kafka?
2. What are the benefits of Kafka?
3. Explain each component of Kafka.
4. Briefly explain Publish-Subscribe Messaging System.
5. Explain Cluster Architecture.
6. What is the role of Zookeeper?
7. Explain Cluster Architecture Components.

*****

9

CASE STUDY: STREAMING REAL TIME DATA (READ TWITTER FEEDS AND EXTRACT THE HASHTAGS)

Unit Structure

9.0 OBJECTIVES

- To stream real time data
- To read Twitter and Extract

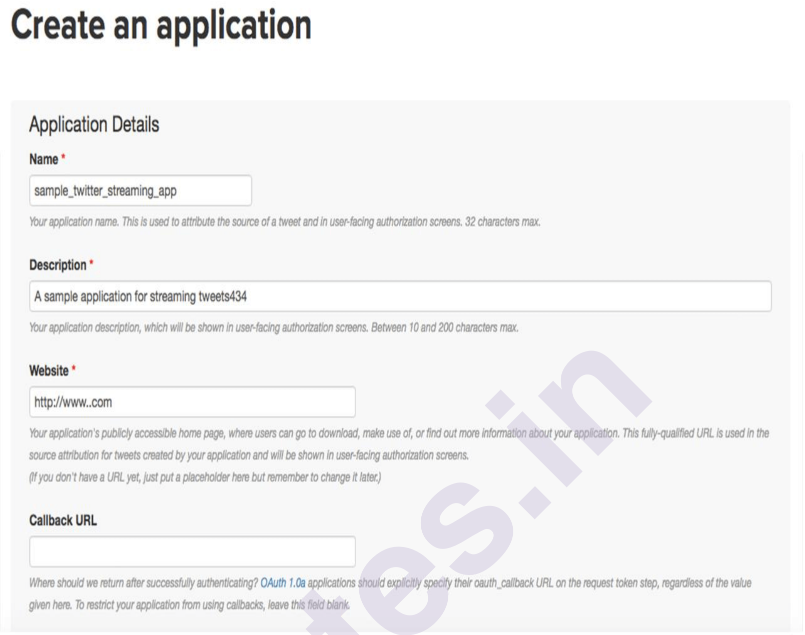
9.1 INTRODUCTION

Social networks are among the biggest sources of data today, and this means they are an extremely valuable asset for marketers, big data specialists, and even individual users like journalists and other professionals. Harnessing the potential of real-time Twitter data is also useful in many time-sensitive business processes. Toptal Freelance Software Engineer Hanee' Medhat explains a simple Python application to leverage the power of Apache Spark, and then use it to read and process tweets to identify trending hashtags.

Nowadays, data is growing and accumulating faster than ever before. Currently, around 90% of all data generated in our world was generated only in the last two years. Due to this staggering growth rate, big data platforms had to adopt radical solutions in order to maintain such huge volumes of data. One of the main sources of data today are social networks. Allow me to demonstrate a real-life example: dealing, analyzing, and extracting insights from social network data in real time using one of the most important big data echo solutions out there—Apache Spark, and Python.

9.2 CREATING OWN CREDENTIALS FOR TWITTER APIS

In order to get tweets from Twitter, you need to register on Twitter Apps by clicking on "Create new app" and then fill the below form click on "Create your Twitter app."



Create an application

Application Details

Name *

sample_twitter_streaming_app

Your application name. This is used to attribute the source of a tweet and in user-facing authorization screens. 32 characters max.

Description *

A sample application for streaming tweets434

Your application description, which will be shown in user-facing authorization screens. Between 10 and 200 characters max.

Website *

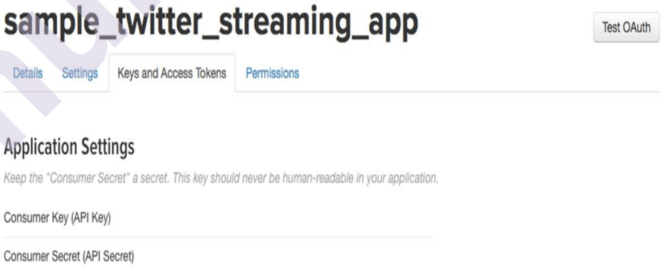
http://www..com

Your application's publicly accessible home page, where users can go to download, make use of, or find out more information about your application. This fully-qualified URL is used in the source attribution for tweets created by your application and will be shown in user-facing authorization screens.
(If you don't have a URL yet, just put a placeholder here but remember to change it later.)

Callback URL

Where should we return after successfully authenticating? OAuth 1.0a applications should explicitly specify their oauth_callback URL on the request token step, regardless of the value given here. To restrict your application from using callbacks, leave this field blank.

Second, go to your newly created app and open the "Keys and Access Tokens" tab. Then click on "Generate my access token."



sample_twitter_streaming_app

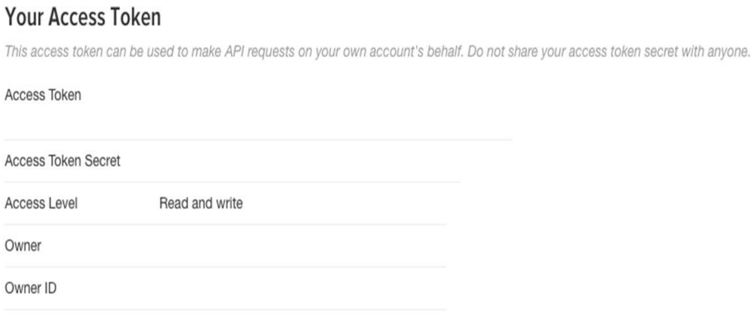
Application Settings

Keep the "Consumer Secret" a secret. This key should never be human-readable in your application.

Consumer Key (API Key)

Consumer Secret (API Secret)

Your new access tokens will appear as below.



Your Access Token

This access token can be used to make API requests on your own account's behalf. Do not share your access token secret with anyone.

Access Token

Access Token Secret

Access Level Read and write

Owner

Owner ID

And now you're ready for the next step.

9.3 BUILDING HE TWITTER HTTP CLIENT

In this step, I’ll show you how to build a simple client that will get the tweets from Twitter API using Python and passes them to the Spark Streaming instance.

First, let’s create a file called twitter_app.py and then we’ll add the code in it together as below.

Import the libraries that we’ll use as below:

```
import socket
import sys
import requests
import requests_oauthlib
import json
```

And add the variables that will be used in OAuth for connecting to Twitter as below:

```
# Replace the values below with yours
ACCESS_TOKEN = 'YOUR_ACCESS_TOKEN'
ACCESS_SECRET = 'YOUR_ACCESS_SECRET'
CONSUMER_KEY = 'YOUR_CONSUMER_KEY'
CONSUMER_SECRET = 'YOUR_CONSUMER_SECRET'
my_auth = requests_oauthlib.OAuth1(CONSUMER_KEY, CONSUMER_SECRET,ACCESS_TOKEN, ACCESS_SECRET)
```

Now, we will create a new function called get_tweets that will call the Twitter API URL and return the response for a stream of tweets.

```
def get_tweets():
        url = 'https://stream.twitter.com/1.1/statuses/filter.json'
        query_data = [('language', 'en'), ('locations', '-130,-20,100,50'),('track','#')]
        query_url = url + '?' + '&'.join([str(t[0]) + '=' + str(t[1]) for t in query_data])
        response = requests.get(query_url, auth=my_auth, stream=True)
        print(query_url, response)
        return response
```

Then, create a function that takes the response from the above one and extracts the tweets’ text from the whole tweets’ JSON object. After that, it sends every tweet to Spark Streaming instance (will be discussed later) through a TCP connection.

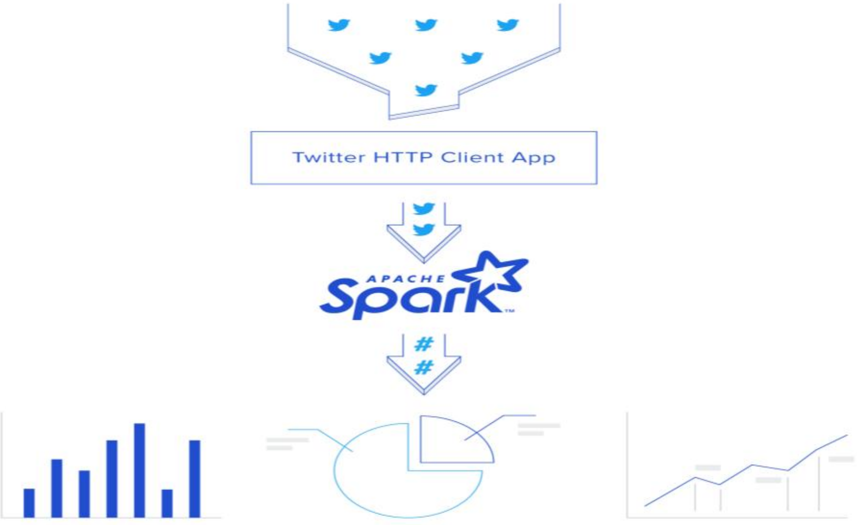
```
def send_tweets_to_spark(http_resp, tcp_connection):
        for line in http_resp.iter_lines():
        try:
        full_tweet = json.loads(line)
        tweet_text = full_tweet['text']
        print("Tweet Text: " + tweet_text)
        print ("-----------------------------------------")
        tcp_connection.send(tweet_text + '\n')
        except:
        e = sys.exc_info()[0]
        print("Error: %s" % e)
```

Now, we'll make the main part which will make the app host socket connections that spark will connect with. We'll configure the IP here to be localhost as all will run on the same machine and the port 9009. Then we'll call the get_tweets method, which we made above, for getting the tweets from Twitter and pass its response along with the socket connection to send_tweets_to_spark for sending the tweets to Spark.

```
TCP_IP = "localhost"
TCP_PORT = 9009
conn = None
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((TCP_IP, TCP_PORT))
s.listen(1)
print("Waiting for TCP connection...")
conn, addr = s.accept()
print("Connected... Starting getting tweets.")
resp = g
et_tweets()
send_tweets_to_spark(resp, conn)
```

9.4 SETTING APACHE SPARK STREAMING UP OUR APPLICATION

Let's build up our Spark streaming app that will do real-time processing for the incoming tweets, extract the hashtags from them, and calculate how many hashtags have been mentioned.



First, we have to create an instance of Spark Context sc, then we created the Streaming Context ssc from sc with a batch interval two seconds that will do the transformation on all streams received every two seconds. Notice we have set the log level to ERROR in order to disable most of the logs that Spark writes.

We defined a checkpoint here in order to allow periodic RDD checkpointing; this is mandatory to be used in our app, as we'll use stateful transformations (will be discussed later in the same section).

Then we define our main DStream dataStream that will connect to the socket server we created before on port 9009 and read the tweets from that port. Each record in the DStream will be a tweet.

```
from pyspark import SparkConf,SparkContext
from pyspark.streaming import StreamingContext
from pyspark.sql import Row,SQLContext
import sys
import requests
# create spark configuration
conf = SparkConf()
conf.setAppName("TwitterStreamApp")
# create spark context with the above configuration
sc = SparkContext(conf=conf)
sc.setLogLevel("ERROR")
# create the Streaming Context from the above spark context with interval size 2 seconds
ssc = StreamingContext(sc, 2)
# setting a checkpoint to allow RDD recovery
```

```
ssc.checkpoint("checkpoint_TwitterApp")
# read data from port 9009
dataStream = ssc.socketTextStream("localhost",9009)
```

Now, we'll define our transformation logic. First we'll split all the tweets into words and put them in words RDD. Then we'll filter only hashtags from all words and map them to pair of (hashtag, 1)and put them in hashtags RDD.

Then we need to calculate how many times the hashtag has been mentioned. We can do that by using the function reduceByKey. This function will calculate how many times the hashtag has been mentioned per each batch, i.e. it will reset the counts in each batch.

In our case, we need to calculate the counts across all the batches, so we'll use another function called updateStateByKey, as this function allows you to maintain the state of RDD while updating it with new data. This way is called Stateful Transformation.

Note that in order to use updateStateByKey, you've got to configure a checkpoint, and that what we have done in the previous step.

```
# split each tweet into words
words = dataStream.flatMap(lambda line: line.split(" "))
# filter the words to get only hashtags, then map each hashtag to be a pair of (hashtag,9)
hashtags = words.filter(lambda w: '#' in w).map(lambda x: (x, 1))
# adding the count of each hashtag to its last count
tags_totals = hashtags.updateStateByKey(aggregate_tags_count)
# do processing for each RDD generated in each interval
tags_totals.foreachRDD(process_rdd)
# start the streaming computation
ssc.start()
# wait for the streaming to finish
ssc.awaitTermination()
```

The updateStateByKey takes a function as a parameter called the update function. It runs on each item in RDD and does the desired logic.

In our case, we've created an update function called aggregate_tags_count that will sum all the new_values for each hashtag and add them to the total_sum that is the sum across all the batches and save the data into tags_totals RDD.

```
def aggregate_tags_count(new_values, total_sum):

        return sum(new_values) + (total_sum or 0)
```

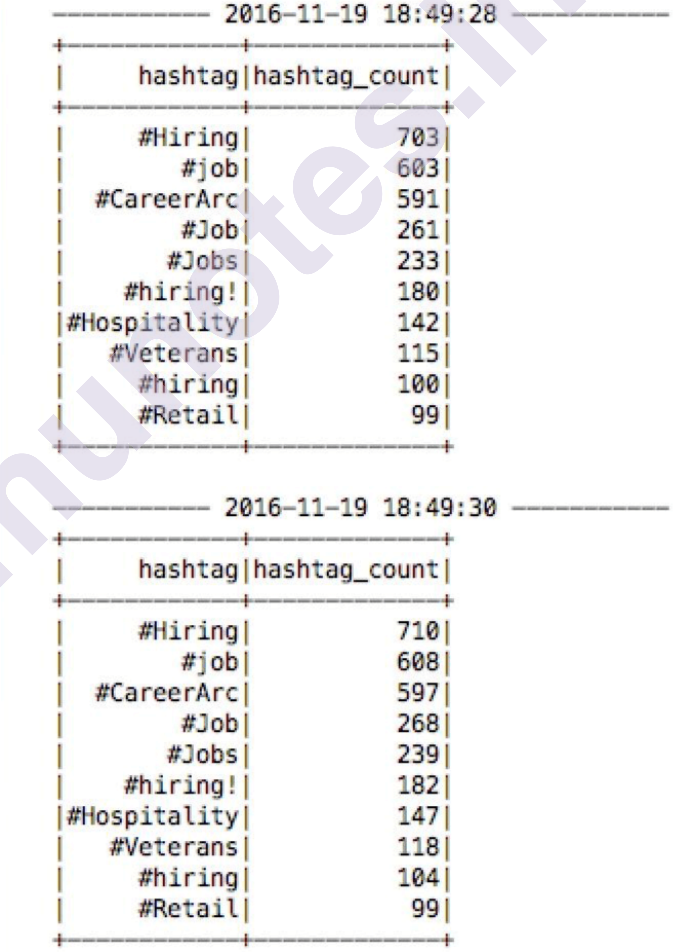
Then we do processing on tags_totals RDD in every batch in order to convert it to temp table using Spark SQL Context and then perform a select statement in order to retrieve the top ten hashtags with their counts and put them into hashtag_counts_df data frame.

```
def get_sql_context_instance(spark_context):

        if ('sqlContextSingletonInstance' not in globals()):

       globals()['sqlContextSingletonInstance'] = SQLContext(spark_context)

        return globals()['sqlContextSingletonInstance']

def process_rdd(time, rdd):

        print("----------- %s -----------" % str(time))

        try:

        # Get spark sql singleton context from the current context

        sql_context = get_sql_context_instance(rdd.context)

        # convert the RDD to Row RDD

        row_rdd = rdd.map(lambda w: Row(hashtag=w[0], hashtag_count=w[1]))

        # create a DF from the Row RDD

        hashtags_df = sql_context.createDataFrame(row_rdd)

        # Register the dataframe as table

        hashtags_df.registerTempTable("hashtags")

        # get the top 10 hashtags from the table using SQL and print them

        hashtag_counts_df = sql_context.sql("select hashtag, hashtag_count from hashtags order by hashtag_count desc limit 10")

        hashtag_counts_df.show()

        # call this method to prepare top 10 hashtags DF and send them

        send_df_to_dashboard(hashtag_counts_df)

        except:

        e = sys.exc_info()[0]

        print("Error: %s" % e)
```

The last step in our Spark application is to send the hashtag_counts_df data frame to the dashboard application. So we'll convert the data frame into two arrays, one for the hashtags and the other for their counts. Then we'll send them to the dashboard application through the REST API.

```
def send_df_to_dashboard(df):
        # extract the hashtags from dataframe and convert them into array
        top_tags = [str(t.hashtag) for t in df.select("hashtag").collect()]
        # extract the counts from dataframe and convert them into array
        tags_count = [p.hashtag_count for p in df.select("hashtag_count").collect()]
        # initialize and send the data through REST API
        url = 'http://localhost:5001/updateData'
        request_data = {'label': str(top_tags), 'data': str(tags_count)}
        response = requests.post(url, data=request_data)
```

Finally, here is a sample output of the Spark Streaming while running and printing the hashtag_counts_df, you’ll notice that the output is printed exactly every two seconds as per the batch intervals.



```
----------- 2016-11-19 18:49:28 -----------
+------------+-------------+
|     hashtag|hashtag_count|
+------------+-------------+
|     #Hiring|          703|
|        #job|          603|
|  #CareerArc|          591|
|        #Job|          261|
|       #Jobs|          233|
|    #hiring!|          180|
|#Hospitality|          142|
|   #Veterans|          115|
|     #hiring|          100|
|     #Retail|           99|
+------------+-------------+

----------- 2016-11-19 18:49:30 -----------
+------------+-------------+
|     hashtag|hashtag_count|
+------------+-------------+
|     #Hiring|          710|
|        #job|          608|
|  #CareerArc|          597|
|        #Job|          268|
|       #Jobs|          239|
|    #hiring!|          182|
|#Hospitality|          147|
|   #Veterans|          118|
|     #hiring|          104|
|     #Retail|           99|
+------------+-------------+
```

9.5 Create a Simple Real-time Dashboard for Representing the Data

Now, we’ll create a simple dashboard application that will be updated in real time by Spark. We’ll build it using Python, Flask, and chart.js

First, let’s create a Python project with the structure seen below and download and add the chart.js file into the static directory.

Then, in the app.py file, we’ll create a function called update_data, which will be called by Spark through the URL http://localhost:5001/updateData in order to update the Global labels and values arrays.

Also, the function refresh_graph_data is created to be called by AJAX request to return the new updated labels and values arrays as JSON. The function get_chart_page will render the chart.htmlpage when called.

```
from flask import Flask,jsonify,request
from flask import render_template
import ast
app = Flask(__name__)
labels = []
values = []
@app.route("/")
def get_chart_page():
        global labels,values
        labels = []
        values = []
        return render_template('chart.html', values=values, labels=labels)
@app.route('/refreshData')
def refresh_graph_data():
        global labels, values
        print("labels now: " + str(labels))
        print("data now: " + str(values))
        return jsonify(sLabel=labels, sData=values)
@app.route('/updateData', methods=['POST'])
def update_data():
        global labels, values
        if not request.form or 'data' not in request.form:
        return "error",400
```

```
    labels = ast.literal_eval(request.form['label'])
    values = ast.literal_eval(request.form['data'])
    print("labels received: " + str(labels))
    print("data received: " + str(values))
    return "success",201
if __name__ == "__main__":
    app.run(host='localhost', port=5001)
```

Now, let's create a simple chart in the chart.html file in order to display the hashtag data and update them in real time. As defined below, we need to import the Chart.js and jquery.min.jsJavaScript libraries.

In the body tag, we have to create a canvas and give it an ID in order to reference it while displaying the chart using JavaScript in the next step.

```
<!DOCTYPE html>
<html>
    <head>
    <meta charset="utf-8"/>
    <title>Top Trending Twitter Hashtags</title>
    <script src='static/Chart.js'></script>
    <script
src="//ajax.googleapis.com/ajax/libs/jquery/1.9.1/jquery.min.js"></script>

    </head>
    <body>
    <h2>Top Trending Twitter Hashtags</h2>
    <div style="width:700px;height=500px">
        <canvas id="chart"></canvas>
    </div>
    </body>
</html>
```

Now, let's construct the chart using the JavaScript code below. First, we get the canvas element, and then we create a new chart object and pass the canvas element to it and define its data object like below.

Note that the data's labels and data are bounded with labels and values variables that are returned while rendering the page when calling a get_chart_page function in the app.py file.

The last remaining part is the function that is configured to do an Ajax request every second and call the URL /refreshData, which will execute refresh_graph_data in app.py and return the new updated data, and then update the char that renders the new data.

```
<script>
  var ctx = document.getElementById("chart");
  var myChart = new Chart(ctx, {
        type: 'horizontalBar',
        data: {
        labels: [{% for item in labels %}
              "{{item}}",
              {% endfor %}],
        datasets: [{
              label: '# of Mentions',
              data: [{% for item in values %}
                   {{item}},
                    {% endfor %}],
              backgroundColor: [
              'rgba(255, 99, 132, 0.2)',
              'rgba(54, 162, 235, 0.2)',
              'rgba(255, 206, 86, 0.2)',
                  'rgba(75, 192, 192, 0.2)',
              'rgba(153, 102, 255, 0.2)',
              'rgba(255, 159, 64, 0.2)',
              'rgba(255, 99, 132, 0.2)',
              'rgba(54, 162, 235, 0.2)',
              'rgba(255, 206, 86, 0.2)',
              'rgba(75, 192, 192, 0.2)',
              'rgba(153, 102, 255, 0.2)'
              ],
              borderColor: [
              'rgba(255,99,132,1)',
              'rgba(54, 162, 235, 1)',
             'rgba(255, 206, 86, 1)',
              'rgba(75, 192, 192, 1)',
              'rgba(153, 102, 255, 1)',
              'rgba(255, 159, 64, 1)',
            'rgba(255,99,132,1)',
            'rgba(54, 162, 235, 1)',
            'rgba(255, 206, 86, 1)',
            'rgba(75, 192, 192, 1)',
            'rgba(153, 102, 255, 1)'
            ],
            borderWidth: 1
      }]
      },
      options: {
      scales: {
              yAxes: [{
            ticks: {
                  beginAtZero:true
            }
            }]
      }
      }
  });
  var src_Labels = [];
  var src_Data = [];
  setInterval(function(){
      $.getJSON('/refreshData', {
      }, function(data) {
      src_Labels = data.sLabel;
      src_Data = data.sData;
      });
      myChart.data.labels = src_Labels;
      myChart.data.datasets[0].data = src_Data;
      myChart.update();
  },1000);
</script>
```

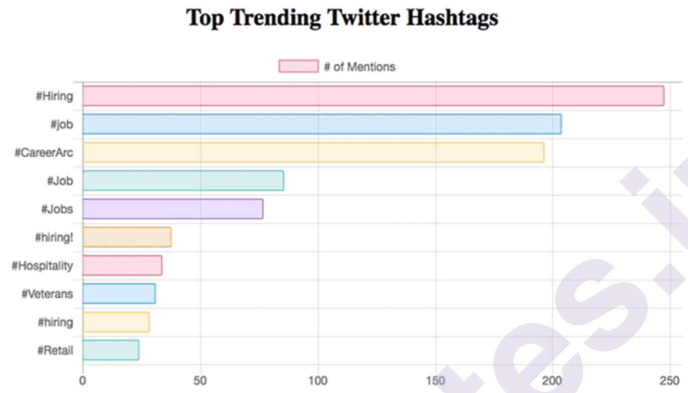
9.6 RUNNING THE APPLICATIONS TOGETHER

Let’s run the three applications in the order below:

1. Twitter App Client.
2. Spark App.
3. Dashboard Web App.

Then you can access the real-time dashboard using the URL <http://localhost:5001/>

Now, you can see your chart being updated, as below:



9.7 APACHE STREAMING REAL LIFE USE CASES

We’ve learned how to do simple data analytics on data in real time using Spark Streaming and integrating it directly with a simple dashboard using a RESTful web service. From this example, we can see how powerful Spark is, as it captures a massive stream of data, transforms it, and extracts valuable insights that can be used easily to make decisions in no time. There are many helpful use cases that can be implemented and which can serve different industries, like news or marketing.

News industry example:

We can track the most frequently mentioned hashtags to know what topics people are talking about the most on social media. Also, we can track specific hashtags and their tweets in order to know what people are saying about specific topics or events in the world.

Marketing example:

We can collect the stream of tweets and, by doing sentiment analysis, categorize them and determine people’s interests in order to target them with offers related to their interests.Also, there are a lot of use cases that can be applied specifically for big data analytics and can serve a lot of industries.

9.8 LET US SUM UP

- Harnessing the potential of real-time Twitter data is also useful in many time-sensitive business processes.
- Dealing, analyzing, and extracting insights from social network data in real time using one of the most important big data echo solutions out there—Apache Spark, and Python.
- A simple application that reads online streams from Twitter using Python, then processes the tweets using Apache Spark Streaming to identify hashtags and, finally, returns top trending hashtags and represents this data on a real-time dashboard.
- Simple data analytics on data in real time using Spark Streaming and integrating it directly with a simple dashboard using a RESTful web service

9.9 LIST OF REFERENCES

- Streaming Systems by Tyler Akidau, Slava Chernyak, Reuven Lax
- Streaming Data - Understanding the Real - time Pipeline First Edition (English, Paperback, Andrew G. Psaltis)
- Real-Time Analytics: Techniques to Analyze and Visualize Streaming Data, Byron Ellis

9.10 BIBLIOGRAPHY

- https://www.toptal.com/apache/apache-spark-streaming-twitter
- Streaming Systems by Tyler Akidau, Slava Chernyak, Reuven Lax
- Streaming Data - Understanding the Real - time Pipeline First Edition (English, Paperback, Andrew G. Psaltis)

9.11 UNIT END EXERCISE

1. What is Data Streaming?
2. Explain with the steps how to read Twitter Feeds and Extract the Hashtags.
3. How to create Credentials for Twitter API?
4. How to build Twitter HTTP client?
5. How to Create a Simple Real-time Dashboard for Representing the Data?

*****

10

APACHE SPARK

Unit Structure

10.0 OBJECTIVES

- To explore Apache Spark
- To work with RDD in Spark
- To Explore Dataset and Dataframe
- To gain knowledge in Spark SQL

10.1 INTRODUCTION

Apache Spark is a lightning-fast cluster computing designed for fast computation. It was built on top of Hadoop MapReduce and it extends the MapReduce model to efficiently use more types of computations which includes Interactive Queries and Stream Processing.

Industries are using Hadoop extensively to analyze their data sets. The reason is that Hadoop framework is based on a simple programming model (MapReduce) and it enables a computing solution that is scalable, flexible, fault-tolerant and cost effective. Here, the main concern is to maintain speed in processing large datasets in terms of waiting time between queries and waiting time to run the program.

Spark was introduced by Apache Software Foundation for speeding up the Hadoop computational computing software process.

As against a common belief, **Spark is not a modified version of Hadoop** and is not, really, dependent on Hadoop because it has its own cluster management. Hadoop is just one of the ways to implement Spark.

Spark uses Hadoop in two ways – one is **storage** and second is **processing**. Since Spark has its own cluster management computation, it uses Hadoop for storage purpose only.

10.2 SPARK BASICS

Apache Spark:

Apache Spark is a lightning-fast cluster computing technology, designed for fast computation. It is based on Hadoop MapReduce and it extends the MapReduce model to efficiently use it for more types of computations, which includes interactive queries and stream processing. The main feature of Spark is its **in-memory cluster computing** that increases the processing speed of an application.

Spark is designed to cover a wide range of workloads such as batch applications, iterative algorithms, interactive queries and streaming. Apart from supporting all these workload in a respective system, it reduces the management burden of maintaining separate tools.

Evolution of Apache Spark:

Spark is one of Hadoop's sub project developed in 2009 in UC Berkeley's AMPLab by Matei Zaharia. It was Open Sourced in 2010 under a BSD license. It was donated to Apache software foundation in 2013, and now Apache Spark has become a top level Apache project from Feb-2014.

Features of Apache Spark:

Apache Spark has following features.

- **Speed:** Spark helps to run an application in Hadoop cluster, up to 100 times faster in memory, and 10 times faster when running on disk. This is possible by reducing number of read/write operations to disk. It stores the intermediate processing data in memory.
- **Supports multiple languages:** Spark provides built-in APIs in Java, Scala, or Python. Therefore, you can write applications in different languages. Spark comes up with 80 high-level operators for interactive querying.
- Advanced Analytics: Spark not only supports 'Map' and 'reduce'. It also supports SQL queries, Streaming data, Machine learning (ML), and Graph algorithms.

Spark Built on Hadoop:

There are three ways of Spark deployment as explained below.

- **Standalone:** Spark Standalone deployment means Spark occupies the place on top of HDFS(Hadoop Distributed File System) and space is allocated for HDFS, explicitly. Here, Spark and MapReduce will run side by side to cover all spark jobs on cluster.

- **Hadoop Yarn:** Hadoop Yarn deployment means, simply, spark runs on Yarn without any pre-installation or root access required. It helps to integrate Spark into Hadoop ecosystem or Hadoop stack. It allows other components to run on top of stack.
- **Spark in MapReduce (SIMR):** Spark in MapReduce is used to launch spark job in addition to standalone deployment. With SIMR, user can start Spark and uses its shell without any administrative access.

Components of Spark:

Apache Spark Core:

Spark Core is the underlying general execution engine for spark platform that all other functionality is built upon. It provides In-Memory computing and referencing datasets in external storage systems.

Spark SQL:

Spark SQL is a component on top of Spark Core that introduces a new data abstraction called SchemaRDD, which provides support for structured and semi-structured data.

Spark Streaming:

Spark Streaming leverages Spark Core's fast scheduling capability to perform streaming analytics. It ingests data in mini-batches and performs RDD (Resilient Distributed Datasets) transformations on those mini-batches of data.

MLlib (Machine Learning Library):

MLlib is a distributed machine learning framework above Spark because of the distributed memory-based Spark architecture. It is, according to benchmarks, done by the MLlib developers against the Alternating Least Squares (ALS) implementations. Spark MLlib is nine times as fast as the Hadoop disk-based version of **Apache Mahout**(before Mahout gained a Spark interface).

GraphX:

GraphX is a distributed graph-processing framework on top of Spark. It provides an API for expressing graph computation that can model the user-defined graphs by using Pregel abstraction API. It also provides an optimized runtime for this abstraction.

10.3 WORKING WITH RDDS IN SPARK

Resilient Distributed Datasets:

Resilient Distributed Datasets (RDD) is a fundamental data structure of Spark. It is an immutable distributed collection of objects. Each dataset in RDD is divided into logical partitions, which may be computed on different nodes of the cluster. RDDs can contain any type of Python, Java, or Scala objects, including user-defined classes.

Formally, an RDD is a read-only, partitioned collection of records. RDDs can be created through deterministic operations on either data on stable storage or other RDDs. RDD is a fault-tolerant collection of elements that can be operated on in parallel.

There are two ways to create RDDs − **parallelizing** an existing collection in your driver program, or **referencing a dataset** in an external storage system, such as a shared file system, HDFS, HBase, or any data source offering a Hadoop Input Format.

Spark makes use of the concept of RDD to achieve faster and efficient MapReduce operations. Let us first discuss how MapReduce operations take place and why they are not so efficient.

Data Sharing is Slow in MapReduce:

MapReduce is widely adopted for processing and generating large datasets with a parallel, distributed algorithm on a cluster. It allows users to write parallel computations, using a set of high-level operators, without having to worry about work distribution and fault tolerance.

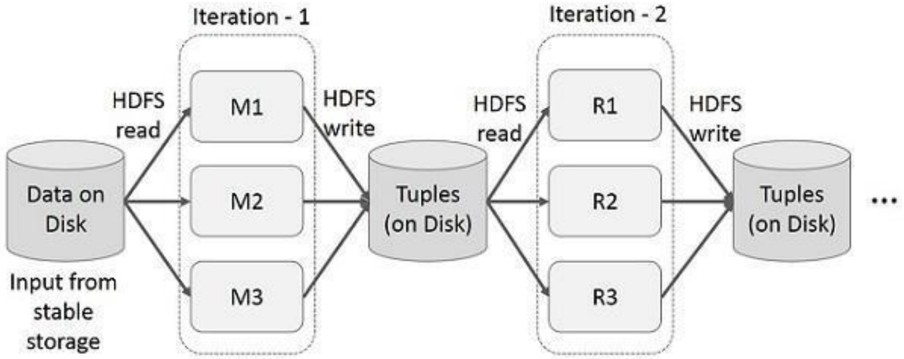
Unfortunately, in most current frameworks, the only way to reuse data between computations (Ex − between two MapReduce jobs) is to write it to an external stable storage system (Ex − HDFS). Although this framework provides numerous abstractions for accessing a cluster's computational resources, users still want more.

Both **Iterative** and **Interactive** applications require faster data sharing across parallel jobs. Data sharing is slow in MapReduce due to **replication, serialization**, and **disk IO**. Regarding storage system, most of the Hadoop applications, they spend more than 90% of the time doing HDFS read-write operations.

Iterative Operations on MapReduce:

Reuse intermediate results across multiple computations in multi-stage applications. The following illustration explains how the current framework works, while doing the iterative operations on MapReduce. This incurs substantial overheads due to data replication, disk I/O, and serialization, which makes the system slow.

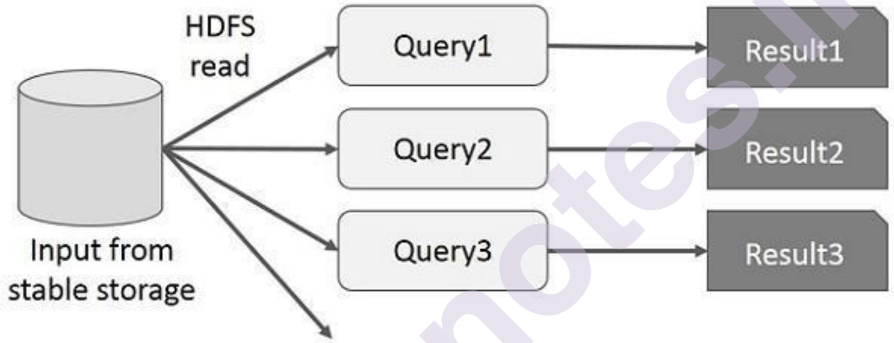
Reuse intermediate results across multiple computations in multi-stage applications. The following illustration explains how the current framework works, while doing the iterative operations on MapReduce. This incurs substantial overheads due to data replication, disk I/O, and serialization, which makes the system slow.



Interactive Operations on MapReduce:

User runs ad-hoc queries on the same subset of data. Each query will do the disk I/O on the stable storage, which can dominate application execution time.

The following illustration explains how the current framework works while doing the interactive queries on MapReduce.



Data Sharing using Spark RDD:

Data sharing is slow in MapReduce due to **replication, serialization**, and **disk IO**. Most of the Hadoop applications, they spend more than 90% of the time doing HDFS read-write operations.

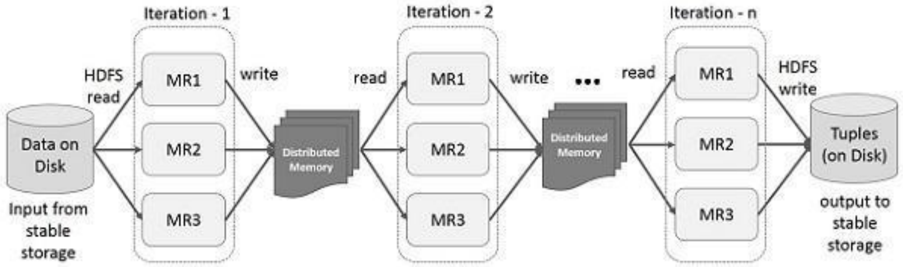
Recognizing this problem, researchers developed a specialized framework called Apache Spark. The key idea of spark is **R**esilient **D**istributed **D**atasets (RDD); it supports in-memory processing computation. This means, it stores the state of memory as an object across the jobs and the object is sharable between those jobs. Data sharing in memory is 10 to 100 times faster than network and Disk.

Let us now try to find out how iterative and interactive operations take place in Spark RDD.

Iterative Operations on Spark RDD:

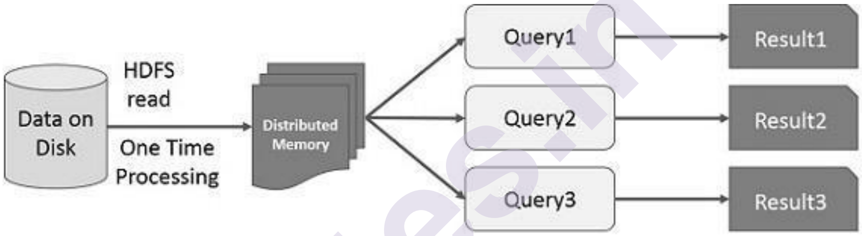
The illustration given below shows the iterative operations on Spark RDD. It will store intermediate results in a distributed memory instead of Stable storage (Disk) and make the system faster.

Note: If the Distributed memory (RAM) is not sufficient to store intermediate results (State of the JOB), then it will store those results on the disk.



Interactive Operations on Spark RDD:

This illustration shows interactive operations on Spark RDD. If different queries are run on the same set of data repeatedly, this particular data can be kept in memory for better execution times.



By default, each transformed RDD may be recomputed each time you run an action on it. However, you may also **persist** an RDD in memory, in which case Spark will keep the elements around on the cluster for much faster access, the next time you query it. There is also support for persisting RDDs on disk, or replicated across multiple nodes.

10.4 SPARK FRAMEWORK

Apache Spark define is a data processing framework that can quickly perform processing tasks on very large data sets and can also distribute data processing tasks across multiple computers, either on its own or in tandem with other distributed computing tools. These two qualities are key to the worlds of big data and machine learning, which require the marshalling of massive computing power to crunch through large data stores. Spark also takes some of the programming burdens of these tasks off the shoulders of developers with an easy-to-use API that abstracts away much of the grunt work of distributed computing and big data processing.

10.5 AGGREGATING DATA WITH PAIR RDDS

Apache Spark’s Core abstraction is Resilient Distributed Datasets, an acronym for Resilient Distributed Datasets is RDD. Also, a fundamental data structure of Spark. Moreover, Spark RDDs is immutable in nature. As well as the distributed collection of objects. Basically, RDD in spark is designed as each dataset in RDD is divided into logical partitions. Further, we can say here each partition may be computed on different nodes of the cluster. Moreover, Spark RDDs contain user-defined classes.

Paired RDD is a distributed collection of data with the key-value pair. It is a subset of Resilient Distributed Dataset So it has all the features of RDD and some new feature for the key-value pair. There are many transformation operations available for Paired RDD.

These operations on Paired RDD are very useful to solve many use cases that require sorting, grouping, reducing some value/function. Commonly used operations on paired RDD are: groupByKey() reduceByKey() countByKey() join() etc.

Here is an example regarding creating of paired RDD:

```
val pRDD:[(String),(Int)]=sc.textFile(“path_of_your_file”)

.flatMap(line => line.split(” “))
```

Spark Paired RDD Operations:

a. Transformation Operations:

Paired RDD allows the same transformation those are available to standard RDDs. Moreover, here also same rules apply from “passing functions to spark”. Also in Spark, there are tuples available in paired RDDs. Basically, we need to pass functions that operate on tuples, despite on individual elements. Let’s discuss some of the transformation methods below, like

- **groupByKey:**

The groupbykey operation generally groups all the values with the same key.
rdd.groupByKey()

- **reduceByKey(fun):**

Here, the reduceByKey operation generally combines values with the same key.
add.reduceByKey((x, y) => x + y)

- **combineByKey(createCombiner, mergeValue, mergeCombiners, partitioner):**

CombineByKey uses a different result type, then combine those values with the same key.

- **mapValues(func):**

Even without changing the key, mapValues operation applies a function to each value of a paired RDD of spark.
rdd.mapValues(x => x+1)

- **keys():**

Keys() operation generally returns a spark RDD of just the keys. **rdd.keys()**

- **values():**

values() operation generally returns an RDD of just the values. **rdd.values()**

- **sortByKey():**

Similarly, the sortByKey operation generally returns an RDD sorted by the key.

rdd.sortByKey()

b. Action Operations:

As similar as RDD transformations, there are same RDD actions available on spark pair RDD. However, paired RDDs also attains some additional actions of spark. Basically, those leverages the advantage of data which is of keyvalue nature. Let's discuss some of the action methods below, like

- **countByKey():**

Through countByKey operation, we can count the number of elements for each key. **rdd.countByKey()**

- **collectAsMap():**

Here, collectAsMap() operation helps to collect the result as a map to provide easy lookup. **rdd.collectAsMap()**

- **lookup(key):**

Moreover, it returns all values associated with the provided key. **rdd.lookup()**

10.6 WRITING AND DEPLOYING SPARK APPLICATIONS

Spark application, using spark-submit, is a shell command used to deploy the Spark application on a cluster. It uses all respective cluster managers through a uniform interface. Therefore, you do not have to configure your application for each one.

Example:

Let us take the same example of word count, we used before, using shell commands. Here, we consider the same example as a spark application.

Sample Input:

The following text is the input data and the file named is **in.txt**.

```
people are not as beautiful as they look,  as they walk or as they talk.  they are
only as beautiful as they love,  as they care as they share.
```

Look at the following program −

SparkWordCount.scala:

```
import org.apache.spark.SparkContext
import org.apache.spark.SparkContext._
import org.apache.spark._

object SparkWordCount {
  def main(args: Array[String]) {

      val sc = new SparkContext( "local", "Word Count", "/usr/local/spark",
Nil, Map(), Map())

      /* local = master URL; Word Count = application name; */
      /* /usr/local/spark = Spark Home; Nil = jars; Map = environment */
      /* Map = variables to work nodes */
      /*creating an inputRDD to read text file (in.txt) through Spark context*/
      val input = sc.textFile("in.txt")
      /* Transform the inputRDD into countRDD */
val count = input.flatMap(line ⇒ line.split(" "))
      .map(word ⇒ (word, 1))
      .reduceByKey(_ + _)

      /* saveAsTextFile method is an action that effects on the RDD */
      count.saveAsTextFile("outfile")
      System.out.println("OK");
  }
}
```

Save the above program into a file named **SparkWordCount.scala** and place it in a user-defined directory named **spark-application**.

Note: While transforming the inputRDD into countRDD, we are using flatMap() for tokenizing the lines (from text file) into words, map() method for counting the word frequency and reduceByKey() method for counting each word repetition.

Use the following steps to submit this application. Execute all steps in the **spark-application** directory through the terminal.

Step 1: Download Spark Ja:

Spark core jar is required for compilation, therefore, download spark-core_2.10-1.3.0.jar from the following link Spark core jar and move the jar file from download directory to **spark-application** directory.

Step 2: Compile program:

Compile the above program using the command given below. This command should be executed from the spark-application directory. Here, **/usr/local/spark/lib/spark-assembly-1.4.0-hadoop2.6.0.jar** is a Hadoop support jar taken from Spark library.

```
$ scalac -classpath "spark-core_2.10-1.3.0.jar:/usr/local/spark/lib/spark-assembly-1.4.0-hadoop2.6.0.jar" SparkPi.scala
```

Step 3: Create a JAR:

Create a jar file of the spark application using the following command. Here, **wordcount** is the file name for jar file.

```
jar -cvf wordcount.jar SparkWordCount*.class spark-core_2.10-1.3.0.jar/usr/local/spark/lib/spark-assembly-1.4.0-hadoop2.6.0.jar
```

Step 4: Submit spark application:

Submit the spark application using the following command −

```
spark-submit --class SparkWordCount --master local wordcount.jar
```

If it is executed successfully, then you will find the output given below. The **OK** letting in the following output is for user identification and that is the last line of the program. If you carefully read the following output, you will find different things, such as −

- successfully started service 'sparkDriver' on port 42954
- MemoryStore started with capacity 267.3 MB
- Started SparkUI at http://192.168.1.217:4040
- Added JAR file:/home/hadoop/piapplication/count.jar
- ResultStage 1 (saveAsTextFile at SparkPi.scala:11) finished in 0.566 s
- Stopped Spark web UI at http://192.168.1.217:4040
- MemoryStore cleared

```
15/07/08 13:56:04 INFO Slf4jLogger: Slf4jLogger started  15/07/08
13:56:04 INFO Utils: Successfully started service 'sparkDriver' on port
42954.  15/07/08 13:56:04 INFO Remoting: Remoting started; listening on
addresses :[akka.tcp://sparkDriver@192.168.1.217:42954]  15/07/08
13:56:04 INFO MemoryStore: MemoryStore started with capacity 267.3
MB  15/07/08 13:56:05 INFO HttpServer: Starting HTTP Server  15/07/08
13:56:05 INFO Utils: Successfully started service 'HTTP file server' on
port 56707.  15/07/08 13:56:06 INFO SparkUI: Started SparkUI at
http://192.168.1.217:4040  15/07/08 13:56:07 INFO SparkContext: Added
JAR file:/home/hadoop/piapplication/count.jar at
http://192.168.1.217:56707/jars/count.jar with timestamp 1436343967029
15/07/08 13:56:11 INFO Executor: Adding file:/tmp/spark-45a07b83-
42ed-42b3b2c2-823d8d99c5af/userFiles-df4f4c20-a368-4cdd-a2a7-
39ed45eb30cf/count.jar to class loader  15/07/08 13:56:11 INFO
HadoopRDD: Input split: file:/home/hadoop/piapplication/in.txt:0+54
15/07/08 13:56:12 INFO Executor: Finished task 0.0 in stage 0.0 (TID 0).
2001 bytes result sent to driver  (MapPartitionsRDD[5] at saveAsTextFile
at SparkPi.scala:11), which is now runnable  15/07/08 13:56:12 INFO
DAGScheduler: Submitting 1 missing tasks from ResultStage 1
(MapPartitionsRDD[5] at saveAsTextFile at SparkPi.scala:11)  15/07/08
13:56:13 INFO DAGScheduler: ResultStage 1 (saveAsTextFile at
SparkPi.scala:11) finished in 0.566 s  15/07/08 13:56:13 INFO
DAGScheduler: Job 0 finished: saveAsTextFile at SparkPi.scala:11, took
2.892996 s OK  15/07/08 13:56:13 INFO SparkContext: Invoking stop()
from shutdown hook  15/07/08 13:56:13 INFO SparkUI: Stopped Spark
web UI at http://192.168.1.217:4040  15/07/08 13:56:13 INFO
DAGScheduler: Stopping DAGScheduler  15/07/08 13:56:14 INFO
MapOutputTrackerMasterEndpoint: MapOutputTrackerMasterEndpoint
stopped!  15/07/08 13:56:14 INFO Utils: path = /tmp/spark-45a07b83-
42ed-42b3-b2c2823d8d99c5af/blockmgr-ccdda9e3-24f6-491b-b509-
3d15a9e05818, already present as root for deletion.  15/07/08 13:56:14
INFO MemoryStore: MemoryStore cleared  15/07/08 13:56:14 INFO
BlockManager: BlockManager stopped  15/07/08 13:56:14 INFO
BlockManagerMaster: BlockManagerMaster stopped  15/07/08 13:56:14
INFO SparkContext: Successfully stopped SparkContext  15/07/08
13:56:14 INFO Utils: Shutdown hook called  15/07/08 13:56:14 INFO
Utils: Deleting directory /tmp/spark-45a07b83-42ed-42b3b2c2-
823d8d99c5af  15/07/08 13:56:14 INFO
OutputCommitCoordinator$OutputCommitCoordinatorEndpoint:
OutputCommitCoordinator stopped!
```

Step 5: Checking output:

After successful execution of the program, you will find the directory named **outfile** in the spark-application directory.

The following commands are used for opening and checking the list of files in the outfile directory.

```
$ cd outfile  $ ls  Part-00000 part-00001 _SUCCESS
```

The commands for checking output in **part-00000** file are −

```
$ cat part-00000 (people,1) (are,2) (not,1) (as,8) (beautiful,2) (they, 7)
(look,1)
```

The commands for checking output in part-00001 file are −

```
$ cat part-00001 (walk, 1) (or, 1) (talk, 1) (only, 1) (love, 1) (care, 1)
(share, 1)
```

Go through the following section to know more about the ‘spark-submit’ command.

Spark-submit Syntax

```
spark-submit [options] <app jar | python file> [app arguments]
```

Options:

The given below describes a list of options :

Spark contains two different types of shared variables − one is broadcast variables and second is accumulators.

- Broadcast variables − used to efficiently, distribute large values.
- Accumulators − used to aggregate the information of particular collection.

Numeric RDD Operations:

- Spark allows you to do different operations on numeric data, using one of the predefined API methods. Spark’s numeric operations are implemented with a streaming algorithm that allows building the model, one element at a time.
- These operations are computed and returned as a **StatusCounter** object by calling **status()** method.
- The following is a list of numeric methods available in **StatusCounter**.

S.No	Methods & Meaning
1	**count()** Number of elements in the RDD.
2	**Mean()** Average of the elements in the RDD.
3	**Sum()** Total value of the elements in the RDD.

4	**Max()** Maximum value among all elements in the RDD.
5	**Min()** Minimum value among all elements in the RDD.
6	**Variance()** Variance of the elements.
7	**tdev()** Standard deviation.

10.7 SPARK SQL AND DATA FRAMES

Spark SQL is a Spark module for 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.

All of the examples on this page use sample data included in the Spark distribution and can be run in the spark-shell, pyspark shell, or sparkRshell.

SQL:

One use of Spark SQL is to execute SQL queries. Spark SQL can also be used to read data from an existing Hive installation. For more on how to configure this feature, please refer to the Hive Tables section. When running SQL from within another programming language the results will be returned as a Dataset/DataFrame. You can also interact with the SQL interface using the command-line or over JDBC/ODBC.

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. Python does not have the support for the Dataset API. But due to Python's dynamic nature, many of the benefits of the Dataset API are already available (i.e. you can access the field of a row by name naturally row.columnName). The case for R is similar.

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.

Understanding Spark SQL & DataFrames:

Spark SQL essentially tries to bridge the gap between the two models we mentioned previously — the relational and procedural models by two major components.

- Spark SQL provides a DataFrame API that can perform relational operations on both external data sources and Spark's built-in distributed collections — at scale!
- To support the a wide variety of diverse data sources and algorithms in big data, Spark SQL introduces a novel extensible optimizer called Catalyst, which makes it easy to add data sources, optimization rules, and data types for advanced analytics such as machine learning.

Essentially, Spark SQL leverages the power of Spark to perform distributed, robust, in-memory computations at massive scale on Big Data. Spark SQL provides state-of-the-art SQL performance, and also maintains compatibility with all existing structures and components supported by **Apache Hive** (a popular Big Data Warehouse framework) including data formats, user-defined functions (UDFs) and the metastore. Besides this, it also helps in ingesting a wide variety of data formats from Big Data sources and enterprice data warehouses like JSON, Hive, Parquet and so on, and perform a combination of relational and procedural operations for more complex, advanced analytics.

Goals:

Let's look at some of the interesting facts about Spark SQL, it's usage, adoption and goals, some of which I will shamelessly once again copy from the excellent and original paper on Relational Data Processing in Spark. Spark SQL was first released in May 2014, and is perhaps now one of the most actively developed components in Spark. Apache Spark is definitely the most active open source project for big data processing, with hundreds of contributors. Besides being just an open-source project, Spark SQL has actually started seeing mainstream industry adoption! It has already been deployed in very large scale environments. An excellent case-study has been mentioned by Facebook where they talk about **'Apache Spark @Scale: A 60 TB+ production use case'** — Here, they were doing data preparation for entity ranking and their Hive jobs used to take several days and had many challenges, but they were able to successfully able to scale and increase performance using Spark.

10.8 LET US SUM UP

- Apache Spark is a lightning-fast cluster computing designed for fast computation.
- Apache Spark has following features like speed, support multiple languages, Advance analytics.
- Apache Spark define is a data processing framework that can quickly perform processing tasks on very large data sets, and can also distribute data processing tasks across multiple computers, either on its own or in tandem with other distributed computing tools.
- Spark SQL provides state-of-the-art SQL performance, and also maintains compatibility with all existing structures and components supported by **Apache Hive** (a popular Big Data Warehouse framework) including data formats, user-defined functions (UDFs) and the metastore.

10.9 LIST OF REFERENCES

- Learning Spark: Lightning – Fast Big Data Analysis
- Fastdata Processing with Spark by Holden Karau
- Expert Hadoop Administration: Managing, Tuning, and Securing Spark, YARN and HDFS by sam R. Alapati

10.10 BIBLIOGRAPHY

- https://www.tutorialspoint.com/apache_spark/apache_spark_rdd.htm
- Learning Spart: Lightning – Fast Big Data Analysis
- Fastdata Processing with Spark by Holden Karau
- https://databricks.com/spark/getting-started-with-apache-spark
- https://spark.apache.org/docs/latest/sql-programming-guide.html
- https://www.edureka.co/blog/spark-sql-tutorial/
- https://opensource.com/article/19/3/apache-spark-and-dataframes-tutorial

10.11 UNIT END EXERCISE

1. Explain Apache Spark.
2. List out the features of Spark.
3. What are the components of Spark?
4. Explain working with RDDs in Spark.
5. Explain Spark Framework.
6. How to deploy Spark Applications?
7. Explain Spark SQL and Data Frames

*****

11

DATA VISUALIZATION

Unit Structure

11.0 OBJECTIVE

Data visualization is the graphical representation of information and data. By using visual elements like charts, graphs, and maps, data visualization tools provide an accessible way to see and understand trends, outliers, and patterns in data. In the world of Big Data, data visualization tools and technologies are essential to analyse massive amounts of information and make data-driven decisions.

11.1 INTRODUCTION

Data visualization is actually a set of data points and information that are represented graphically to make it easy and quick for user to understand. Data visualization is good if it has a clear meaning, purpose, and is very easy to interpret, without requiring context. Tools of data visualization provide an accessible way to see and understand trends, outliers, and patterns in data by using visual effects or elements such as a chart, graphs, and maps.

Characteristics of Effective Graphical Visual:

- It shows or visualizes data very clearly in an understandable manner.
- It encourages viewers to compare different pieces of data.
- It closely integrates statistical and verbal descriptions of data set.
- It grabs our interest, focuses our mind, and keeps our eyes on message as human brain tends to focus on visual data more than written data.
- It also helps in identifying area that needs more attention and improvement.
- Using graphical representation, a story can be told more efficiently. Also, it requires less time to understand picture than it takes to understand textual data.

Categories of Data Visualization:

Data visualization is very critical to market research where both numerical and categorical data can be visualized that helps in an increase in impacts of insights and also helps in reducing risk of analysis paralysis. So, data visualization is categorized into following categories :

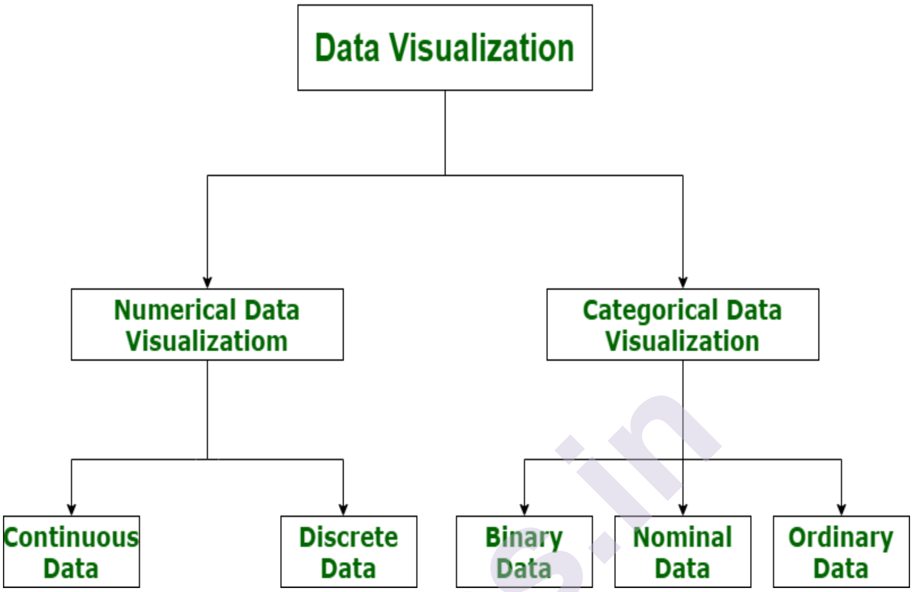


Figure – Categories of Data Visualization

1. Numerical Data:

Numerical data is also known as Quantitative data. Numerical data is any data where data generally represents amount such as height, weight, age of a person, etc. Numerical data visualization is easiest way to visualize data. It is generally used for helping others to digest large data sets and raw numbers in a way that makes it easier to interpret into action. Numerical data is categorized into two categories :

Continuous Data:

It can be narrowed or categorized (Example: Height measurements).

Discrete Data:

This type of data is not “continuous” (Example: Number of cars or children’s a household has).

The type of visualization techniques that are used to represent numerical data visualization is Charts and Numerical Values. Examples are Pie Charts, Bar Charts, Averages, Scorecards, etc.

2. Categorical Data:

Categorical data is also known as Qualitative data. Categorical data is any data where data generally represents groups. It simply consists of categorical variables that are used to represent characteristics such as a person’s ranking, a person’s gender, etc. Categorical data visualization is all about depicting key themes, establishing connections, and lending context. Categorical data is classified into three categories :

Binary Data:

In this, classification is based on positioning (Example: Agrees or Disagrees).

Nominal Data:

In this, classification is based on attributes (Example: Male or Female).

Ordinal Data:

In this, classification is based on ordering of information (Example: Timeline or processes).

The type of visualization techniques that are used to represent categorical data is Graphics, Diagrams, and Flowcharts. Examples are Word clouds, Sentiment Mapping, Venn Diagram, etc.

11.2 EXPLANATION OF DATA VISUALIZATION

What is Data Visualization?

Data visualization is a graphical representation of quantitative information and data by using visual elements like graphs, charts, and maps. Data visualization convert large and small data sets into visuals, which is easy to understand and process for humans. Data visualization tools provide accessible ways to understand outliers, patterns, and trends in the data. In the world of Big Data, the data visualization tools and technologies are required to analyse vast amounts of information.

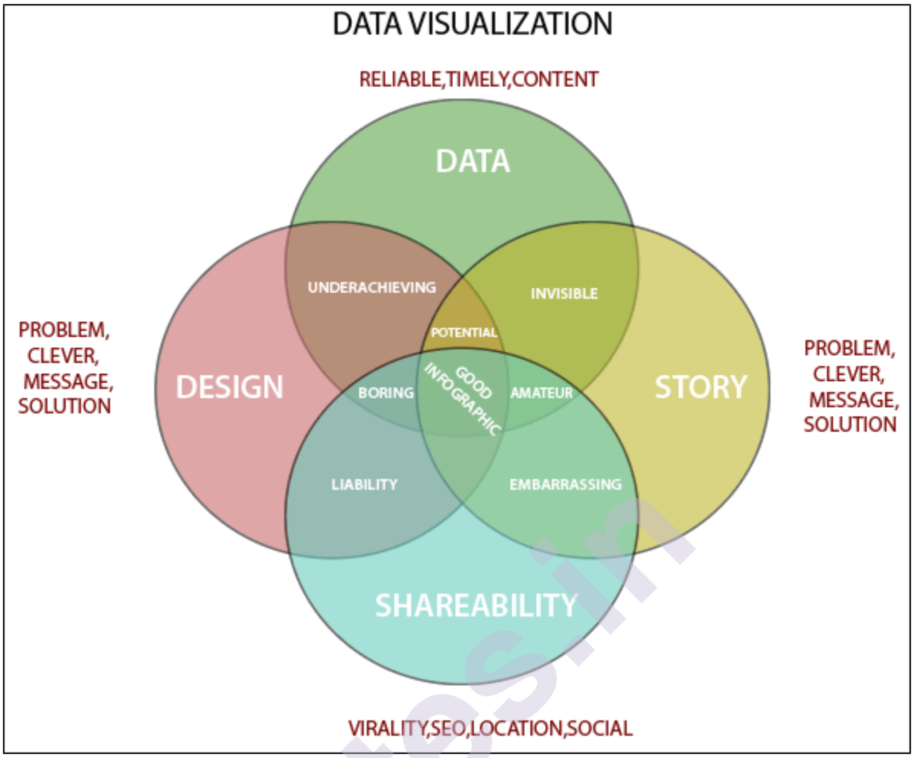
Data visualizations are common in your everyday life, but they always appear in the form of graphs and charts. The combination of multiple visualizations and bits of information are still referred to as Infographics. Data visualizations are used to discover unknown facts and trends. You can see visualizations in the form of line charts to display change over time. Bar and column charts are useful for observing relationships and making comparisons. A pie chart is a great way to show parts-of-a-whole. And maps are the best way to share geographical data visually.

Today's data visualization tools go beyond the charts and graphs used in the Microsoft Excel spreadsheet, which displays the data in more sophisticated ways such as dials and gauges, geographic maps, heat maps, pie chart, and fever chart.

What makes Data Visualization Effective?

Effective data visualization are created by communication, data science, and design collide. Data visualizations did right key insights into complicated data sets into meaningful and natural.

American statistician and Yale professor **Edward Tufte** believe useful data visualizations consist of ?complex ideas communicated with clarity, precision, and efficiency.



To craft an effective data visualization, you need to start with clean data that is well-sourced and complete. After the data is ready to visualize, you need to pick the right chart.

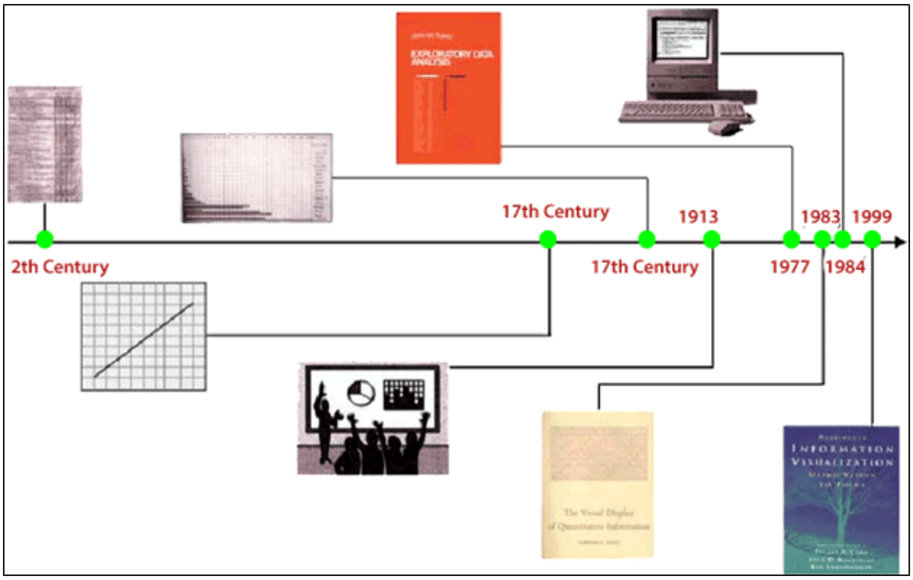
After you have decided the chart type, you need to design and customize your visualization to your liking. Simplicity is essential - you don't want to add any elements that distract from the data.

History of Data Visualization:

The concept of using picture was launched in the 17th century to understand the data from the maps and graphs, and then in the early 1800s, it was reinvented to the pie chart.

Several decades later, one of the most advanced examples of statistical graphics occurred when **Charles Minard** mapped Napoleon's invasion of Russia. The map represents the size of the army and the path of Napoleon's retreat from Moscow - and that information tied to temperature and time scales for a more in-depth understanding of the event.

Computers made it possible to process a large amount of data at lightning-fast speeds. Nowadays, data visualization becomes a fast-evolving blend of art and science that certain to change the corporate landscape over the next few years.


17th Century
1913
1983
1999
2th Century
17th Century
1977
1984
EXPLORATORY DATA ANALYSIS
The Visual Display of Quantitative Information
INFORMATION VISUALIZATION

Importance of Data Visualization:

Data visualization is important because of the processing of information in human brains. Using graphs and charts to visualize a large amount of the complex data sets is more comfortable in comparison to studying the spreadsheet and reports.

Data visualization is an easy and quick way to convey concepts universally. You can experiment with a different outline by making a slight adjustment.

Data visualization has some more specialties such as:

- Data visualization can identify areas that need improvement or modifications.
- Data visualization can clarify which factor influence customer behaviour.
- Data visualization helps you to understand which products to place where.
- Data visualization can predict sales volumes.

Data visualization tools have been necessary for democratizing data, analytics, and making data-driven perception available to workers throughout an organization. They are easy to operate in comparison to earlier versions of BI software or traditional statistical analysis software. This guide to a rise in lines of business implementing data visualization tools on their own, without support from IT.

Why Use Data Visualization?

1. To make easier in understand and remember.
2. To discover unknown facts, outliers, and trends.

3. To visualize relationships and patterns quickly.

4. To ask a better question and make better decisions.

5. To competitive analyse.

6. To improve insights.

11.3 CHALLENGES OF BIG DATA VISUALIZATION

The challenges in Big Data are the real implementation hurdles. These require immediate attention and need to be handled because if not handled then the failure of the technology may take place which can also lead to some unpleasant result. Big data challenges include the storing, analysing the extremely large and fast-growing data.

Some of the Big Data challenges are:

1. Sharing and Accessing Data:

- Perhaps the most frequent challenge in big data efforts is the inaccessibility of data sets from external sources.
- Sharing data can cause substantial challenges.
- It includes the need for inter and intra- institutional legal documents.
- Accessing data from public repositories leads to multiple difficulties.
- It is necessary for the data to be available in an accurate, complete and timely manner because if data in the company's information system is to be used to make accurate decisions in time then it becomes necessary for data to be available in this manner.

2. Privacy and Security:

- It is another most important challenge with Big Data. This challenge includes sensitive, conceptual, technical as well as legal significance.
- Most of the organizations are unable to maintain regular checks due to large amounts of data generation. However, it should be necessary to perform security checks and observation in real time because it is most beneficial.
- There is some information of a person which when combined with external large data may lead to some facts of a person which may be secretive and he might not want the owner to know this information about that person.
- Some of the organization collects information of the people in order to add value to their business. This is done by making insights into their lives that they're unaware of.

3. Analytical Challenges:

- There are some huge analytical challenges in big data which arise some main challenges questions like how to deal with a problem if data volume gets too large?
- Or how to find out the important data points?
- Or how to use data to the best advantage?
- These large amount of data on which these type of analysis is to be done can be structured (organized data), semi-structured (Semi-organized data) or unstructured (unorganized data). There are two techniques through which decision making can be done:
- Either incorporate massive data volumes in the analysis.
- Or determine upfront which Big data is relevant.

4. Technical challenges:

Quality of data:

- When there is a collection of a large amount of data and storage of this data, it comes at a cost. Big companies, business leaders and IT leaders always want large data storage.
- For better results and conclusions, Big data rather than having irrelevant data, focuses on quality data storage.
- This further arise a question that how it can be ensured that data is relevant, how much data would be enough for decision making and whether the stored data is accurate or not.

Fault tolerance:

- Fault tolerance is another technical challenge and fault tolerance computing is extremely hard, involving intricate algorithms.
- Nowadays some of the new technologies like cloud computing and big data always intended that whenever the failure occurs the damage done should be within the acceptable threshold that is the whole task should not begin from the scratch.

Scalability:

- Big data projects can grow and evolve rapidly. The scalability issue of Big Data has lead towards cloud computing.
- It leads to various challenges like how to run and execute various jobs so that goal of each workload can be achieved cost-effectively.
- It also requires dealing with the system failures in an efficient manner. This leads to a big question again that what kinds of storage devices are to be used.

11.4 APPROACHES TO BIG DATA VISUALIZATION

Data visualization is used in many areas to model complex events and visualize phenomena that cannot be observed directly, such as weather patterns, medical conditions or mathematical relationships. Here we review basic data visualization tools and techniques.

By SciForce:

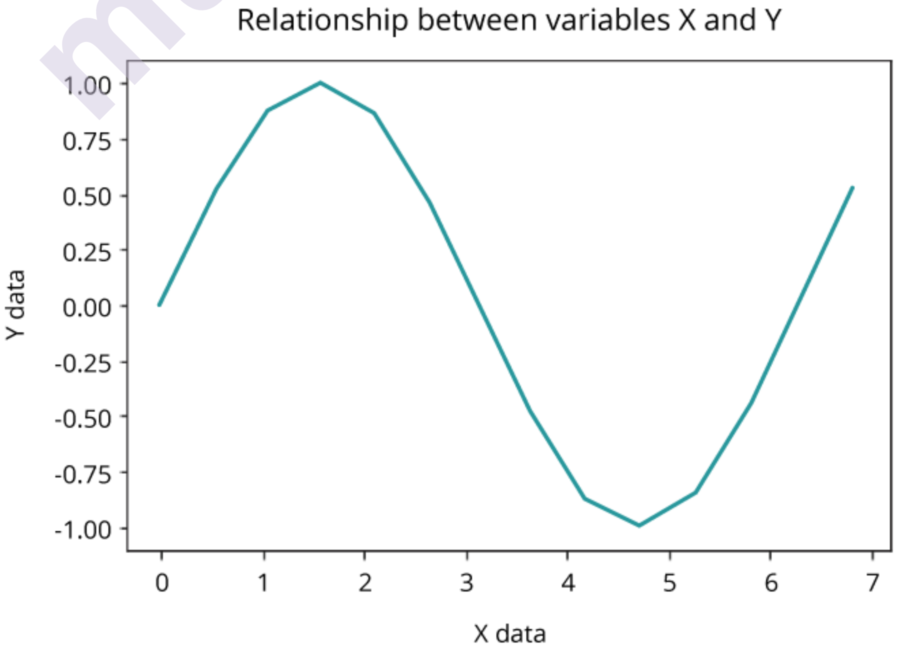
Researchers agree that vision is our dominant sense: 80–85% of information we perceive, learn or process is mediated through vision. It is even more so when we are trying to understand and interpret data or when we are looking for relationships among hundreds or thousands of variables to determine their relative importance. One of the most effective ways to discern important relationships is through advanced analysis and easy-to-understand visualizations.

Data visualization is applied in practically every field of knowledge. Scientists in various disciplines use computer techniques to model complex events and visualize phenomena that cannot be observed directly, such as weather patterns, medical conditions or mathematical relationships.

Data visualization provides an important suite of tools and techniques for gaining a qualitative understanding. The basic techniques are the following plots:

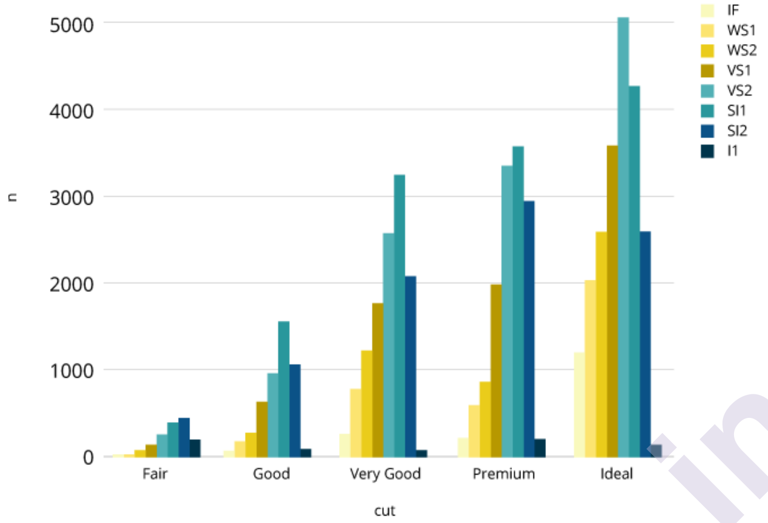
Line Plot:

The simplest technique, a line plot is used to plot the relationship or dependence of one variable on another. To plot the relationship between the two variables, we can simply call the plot function.



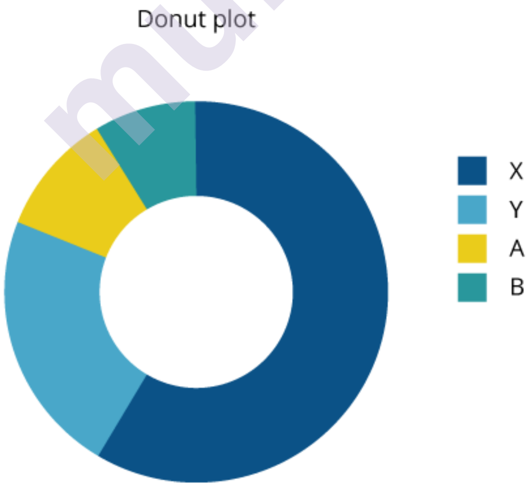
Bar Chart:

Bar charts are used for comparing the quantities of different categories or groups. Values of a category are represented with the help of bars and they can be configured with vertical or horizontal bars, with the length or height of each bar representing the value.



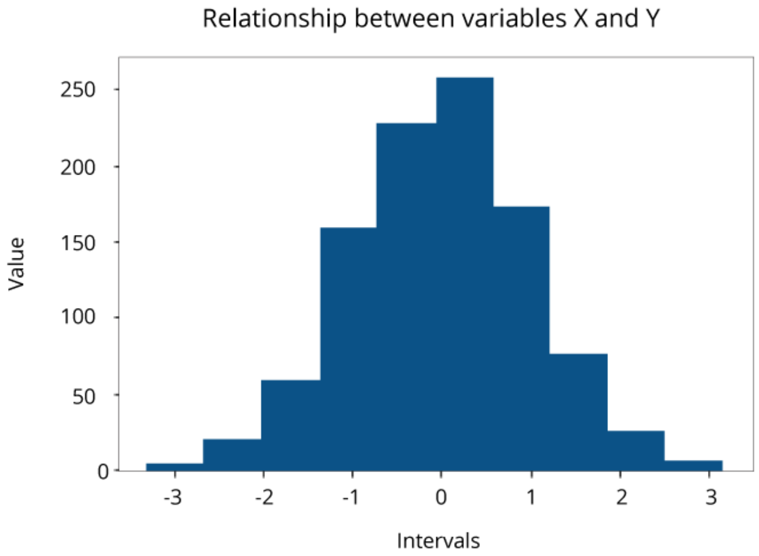
Pie and Donut Charts:

There is much debate around the value of pie and donut charts. As a rule, they are used to compare the parts of a whole and are most effective when there are limited components and when text and percentages are included to describe the content. However, they can be difficult to interpret because the human eye has a hard time estimating areas and comparing visual angles.



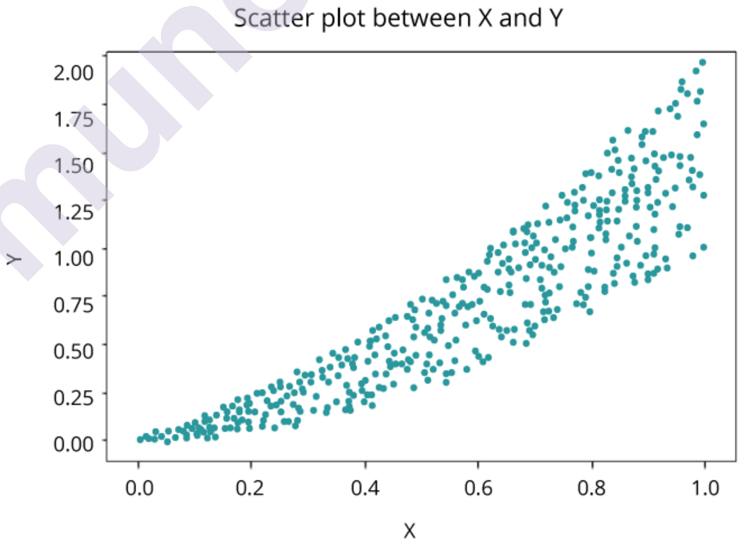
Histogram Plot:

A histogram, representing the distribution of a continuous variable over a given interval or period of time, is one of the most frequently used data visualization techniques in machine learning. It plots the data by chunking it into intervals called 'bins'. It is used to inspect the underlying frequency distribution, outliers, skewness, and so on.



Scatter Plot:

Another common visualization techniques is a scatter plot that is a two-dimensional plot representing the joint variation of two data items. Each marker (symbols such as dots, squares and plus signs) represents an observation. The marker position indicates the value for each observation. When you assign more than two measures, a scatter plot matrix is produced that is a series of scatter plots displaying every possible pairing of the measures that are assigned to the visualization. Scatter plots are used for examining the relationship, or correlations, between X and Y variables.

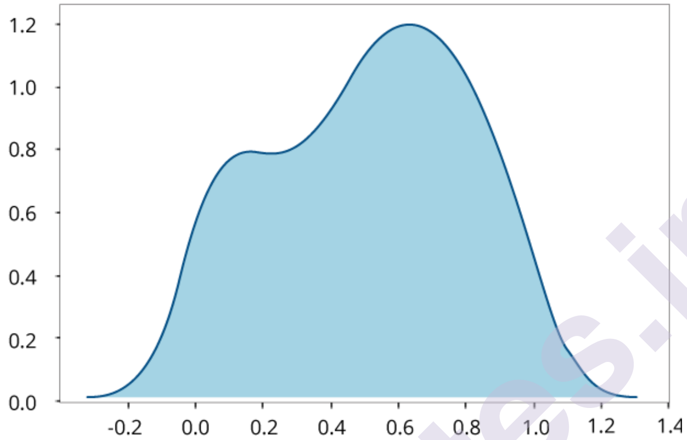


Visualizing Big Data:

Today, organizations generate and collect data each minute. The huge amount of generated data, known as Big Data, brings new challenges to visualization because of the speed, size and diversity of information that must be considered. The volume, variety and velocity of such data requires from an organization to leave its comfort zone technologically to derive intelligence for effective decisions. New and more sophisticated visualization techniques based on core fundamentals of data analysis take into account not only the cardinality, but also the structure and the origin of such data.

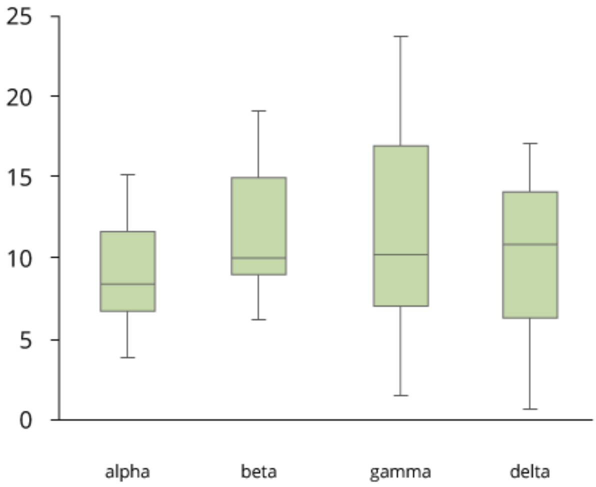
Kernel Density Estimation for Non-Parametric Data:

If we have no knowledge about the population and the underlying distribution of data, such data is called non-parametric and is best visualized with the help of Kernel Density Function that represents the probability distribution function of a random variable. It is used when the parametric distribution of the data doesn't make much sense, and you want to avoid making assumptions about the data.



Box and Whisker Plot for Large Data:

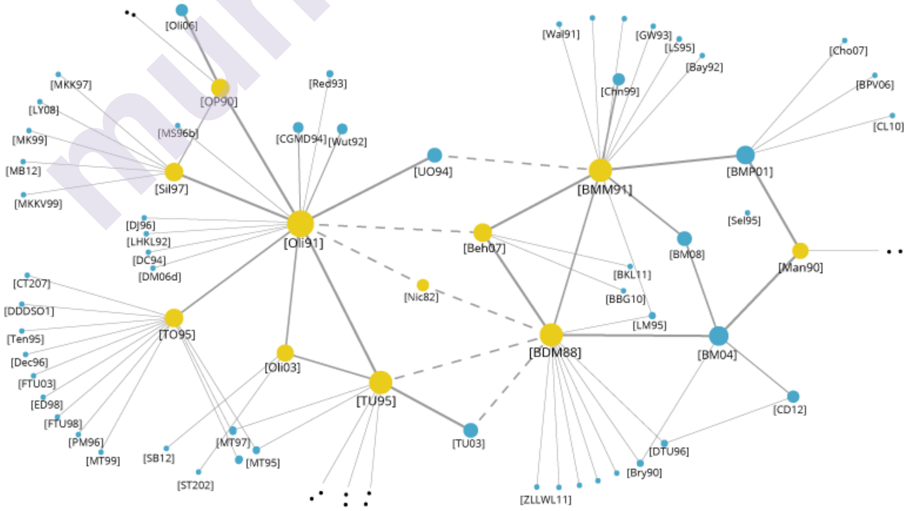
A binned box plot with whiskers shows the distribution of large data and easily see outliers. In its essence, it is a graphical display of five statistics (the minimum, lower quartile, median, upper quartile and maximum) that summarizes the distribution of a set of data. The lower quartile (25th percentile) is represented by the lower edge of the box, and the upper quartile (75th percentile) is represented by the upper edge of the box. The median (50th percentile) is represented by a central line that divides the box into sections. Extreme values are represented by whiskers that extend out from the edges of the box. Box plots are often used to understand the outliers in the data.



Word Clouds and Network Diagrams for Unstructured Data:

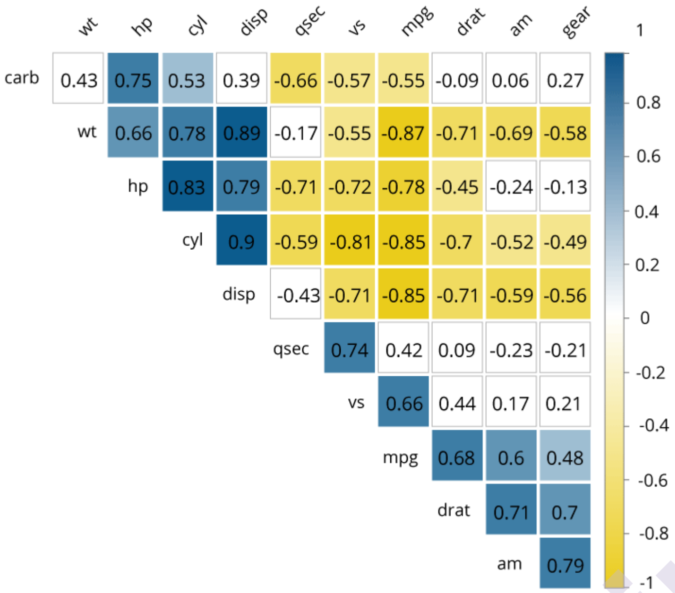
The variety of big data brings challenges because semistructured and unstructured data require new visualization techniques. A word cloud visual represents the frequency of a word within a body of text with its relative size in the cloud. This technique is used on unstructured data as a way to display high- or low-frequency words.

Another visualization technique that can be used for semistructured or unstructured data is the network diagram. Network diagrams represent relationships as nodes (individual actors within the network) and ties (relationships between the individuals). They are used in many applications, for example for analysis of social networks or mapping product sales across geographic areas.



Correlation Matrices:

A correlation matrix allows quick identification of relationships between variables by combining big data and fast response times. Basically, a correlation matrix is a table showing correlation coefficients between variables: Each cell in the table represents the relationship between two variables. Correlation matrices are used as a way to summarize data, as an input into a more advanced analysis, and as a diagnostic for advanced analyses.



Data visualization may become a valuable addition to any presentation and the quickest path to understanding your data. Besides, the process of visualizing data can be both enjoyable and challenging. However, with the many techniques available, it is easy to end up presenting the information using a wrong tool. To choose the most appropriate visualization technique you need to understand the data, its type and composition, what information you are trying to convey to your audience, and how viewers process visual information. Sometimes, a simple line plot can do the task saving time and effort spent on trying to plot the data using advanced Big Data techniques. Understand your data—and it will open its hidden values to you.

*****

12

D3 AND BIG DATA

Unit Structure

12.0 INTRODUCTION

D3 is behind nearly all the most innovative and exciting information visualization on the web today. D3 stands for data-driven documents. It's a brand name, but also a class of applications that have been offered on the web in one form or another for years. In my career, I've made many things that could be considered data-driven documents. These include everything from one-off dynamic maps or social network diagrams to robust visual explorations of time and place. You'll be using D3 whether you're building data visualization prototypes for research or big data dashboards at the top tech companies.

Data visualization is the presentation of data in a pictorial or graphical format. The primary goal of data visualization is to communicate information clearly and efficiently via statistical graphics, plots and information graphics.

Data visualization helps us to communicate our insights quickly and effectively. Any type of data, which is represented by a visualization allows users to compare the data, generate analytic reports, understand patterns and thus helps them to take the decision. Data visualizations can be interactive, so that users analyze specific data in the chart. Well, Data visualizations can be developed and integrated in regular websites and even mobile applications using different JavaScript frameworks.

12.1 GETTING STARTED WITH D3

What is D3.js?

D3 is behind nearly all the most innovative and exciting information visualization on the web today. D3 stands for data-driven documents. It's a brand name, but also a class of applications that have been offered on the web in one form or another for years. In my career, I've made many things that could be considered data-driven documents. These include everything from one-off dynamic maps or social network diagrams to robust visual explorations of time and place. You'll be using D3 whether you're building data visualization prototypes for research or big data dashboards at the top tech companies.

D3.js was created to fill a pressing need for web-accessible, sophisticated data visualization. Let's say your company has used Business Intelligence tools for a while, but they don’t show you the kind of patterns in the data that your team needs. You need to build a custom dashboard that shows exactly how your customers are behaving, tailored for your specific domain. That dashboard needs to be fast, interactive, and shareable around the organization. You’re going to use D3 for that.

D3.js’s creator, Mike Bostock, originally created D3 to take advantage of emerging web standards, which, as he puts it, “avoids proprietary representation and affords extraordinary flexibility, exposing the full capabilities of web standards such as CSS3, HTML5, and SVG” (http://d3js.org). D3.js version 4, the latest iteration of this popular library, continues this trend by modularizing the various pieces of D3 to make it more useful in modern application development.

D3 provides developers with the ability to create rich interactive and animated content based on data and tie that content to existing web page elements. It gives you the tools to create high-performance data dashboards and sophisticated data visualization, and to dynamically update traditional web content.

Your first D3 app:

Throughout this chapter, you’ve seen various lines of code and the effect of those lines of code on the growing d3ia.html sample page you’ve been building. But I’ve avoided explaining the code in too much detail so that you could concentrate on the principles at work in D3. It’s simple to build an application from scratch that uses D3 to create and modify elements. Let’s put it all together and see how it works. First, let’s start with a clean HTML page that doesn’t have any defined styles or existing divs, as shown in the following listing.

Listing 1.6. A simple webpage:

```
<!doctype html>
<html>
<head>
  <script src="d3.v4.min.js"></script>
</head>
<body>
</body>
</html>
```

Hello world with divs:

We can use D3 as an abstraction layer for adding traditional content to the page. Although we can write JavaScript inside our .html file or in its own .js file, let’s put code in the console and see how it works. Later, we’ll focus on the various commands in more detail for layouts and interfaces. We can get started with a piece of code that uses D3 to write to our web page, as in the next listing.

Listing 1.7. Using d3.select to set style and HTML content:

```
d3.select("body").append("div")
  .style("border", "1px black solid")
  .html("hello world");
```

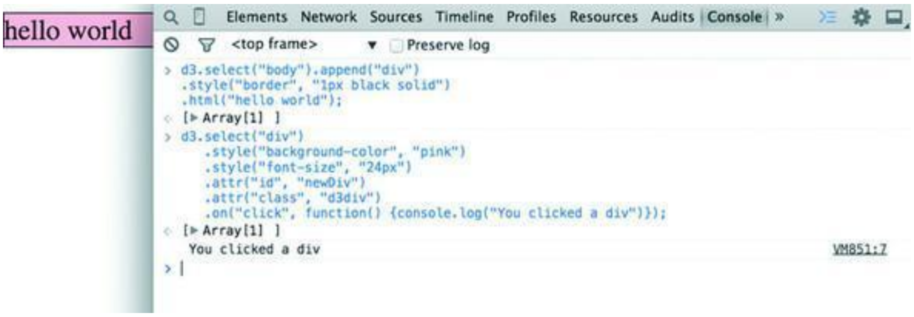
We can adjust the element on the page and give it interactivity with the inclusion of the .on() function, show in the next listing.

Listing 1.8. Using d3.select to set attributes and event listeners:

```
d3.select("div")
  .style("background-color", "pink")
  .style("font-size", "24px")
  .attr("id", "newDiv")
  .attr("class", "d3div")
  .on("click", () => {console.log("You clicked a div")});
```

The .on() function allows us to create an event listener for the currently selected element or set of elements. It accepts the variety of events that can happen to an element, such as click, mouseover, mouseout, and so on. If you click your div, you’ll notice that it gives a response in your console, as shown in figure 1.32.

Figure 1.32. Using console.log(), you can test to see if an event is properly firing. Here you create a <div> and assign an onclick event handler using the .on() syntax. When you click that element and fire the event, the action is noted in the console.



```
> d3.select("body").append("div")
  .style("border", "1px black solid")
  .html("hello world");
< [▶ Array[1] ]
> d3.select("div")
     .style("background-color", "pink")
     .style("font-size", "24px")
     .attr("id", "newDiv")
     .attr("class", "d3div")
     .on("click", function() {console.log("You clicked a div")});
< [▶ Array[1] ]
   You clicked a div                                                    VM851:7
> |
```

Hello World with circles:

You didn’t pick up this book to learn how to add divs to a web page, but you likely want to deal with graphics like lines and circles. To append shapes to a page with D3, you need to have an SVG canvas element somewhere in your page’s DOM. You could either add this SVG canvas to the page as you write the HTML, or you could append it using the D3 syntax you’ve learned:

```
d3.select("body").append("svg");
```

Let’s adjust our d3ia.html page to start with an SVG canvas, as shown in the following listing.

Listing 1.9. A simple web page with an SVG canvas:

```
<html>
<head>
  <script src="d3.v4.min.js"></script>
</head>
<body>
  <div id="vizcontainer">
  <svg style="width:500px;height:500px;border:1px lightgray solid;" />
  </div>
</body>
</html>
```

After we have an SVG canvas on our page, we can append various shapes to it using the same select() and append() syntax we've been using for <div> elements, as shown in the following listing.

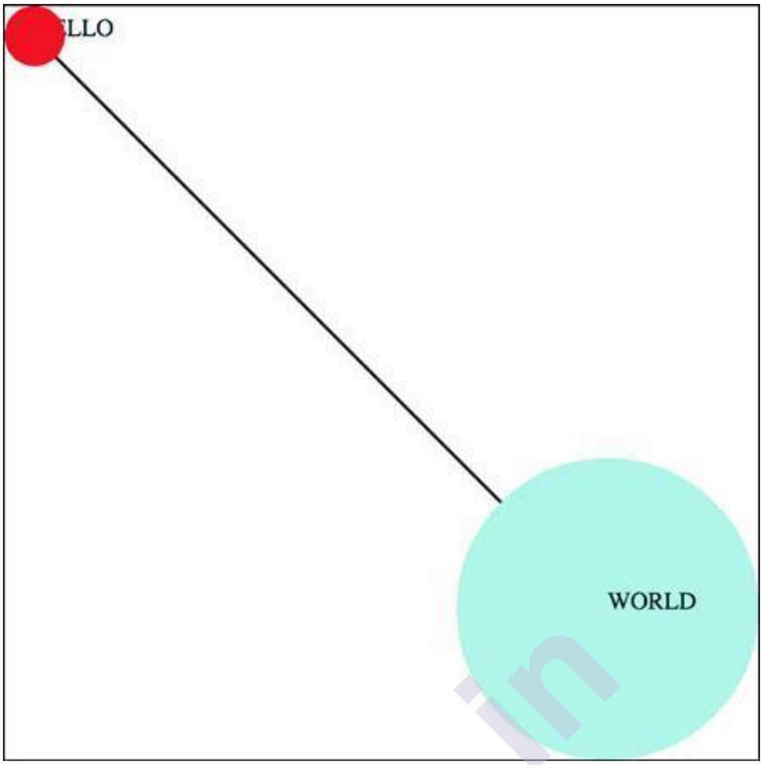
Listing 1.10. Creating lines and circles with select and append:

```
d3.select("svg")
  .append("line")
  .attr("x1", 20)
```

```
 .attr("y1", 20)
 .attr("x2",400)
 .attr("y2",400)
 .style("stroke", "black")
 .style("stroke-width","2px");
d3.select("svg")
 .append("text")
 .attr("x",20)
 .attr("y",20)
 .text("HELLO");
d3.select("svg")
 .append("circle")
 .attr("r", 20)
 .attr("cx",20)
 .attr("cy",20)
 .style("fill","red");
d3.select("svg")
 .append("circle")
 .attr("r", 100)
 .attr("cx",400)
 .attr("cy",400)
 .style("fill","lightblue");
d3.select("svg")
 .append("text")
 .attr("x",400)
 .attr("y",400)
 .text("WORLD");
```

Notice that your circles are drawn over the line, and the text is drawn above or below the circle, depending on the order in which you run your commands, as you can see in figure 1.33. This happened because the draw order of SVG is based on its DOM order. Later you'll learn methods to adjust that order.

Below figure, The result of running listing 1.10 in the console is the creation of two circles, a line, and two text elements. The order in which these elements are drawn results in the first label covered by the circle drawn later.



Why Do We Need D3.js?

D3.js is one of the premier framework when compare to other libraries. This is because it works on the web and its data visualizations are par excellence. Another reason it has worked so well is owing to its flexibility. Since it works seamlessly with the existing web technologies and can manipulate any part of the document object model, it is as flexible as the **Client Side Web Technology Stack** (HTML, CSS, and SVG). It has a great community support and is easier to learn.

D3.js Features:

D3.js is one of the best data visualization framework and it can be used to generate simple as well as complex visualizations along with user interaction and transition effects. Some of its salient features are listed below:

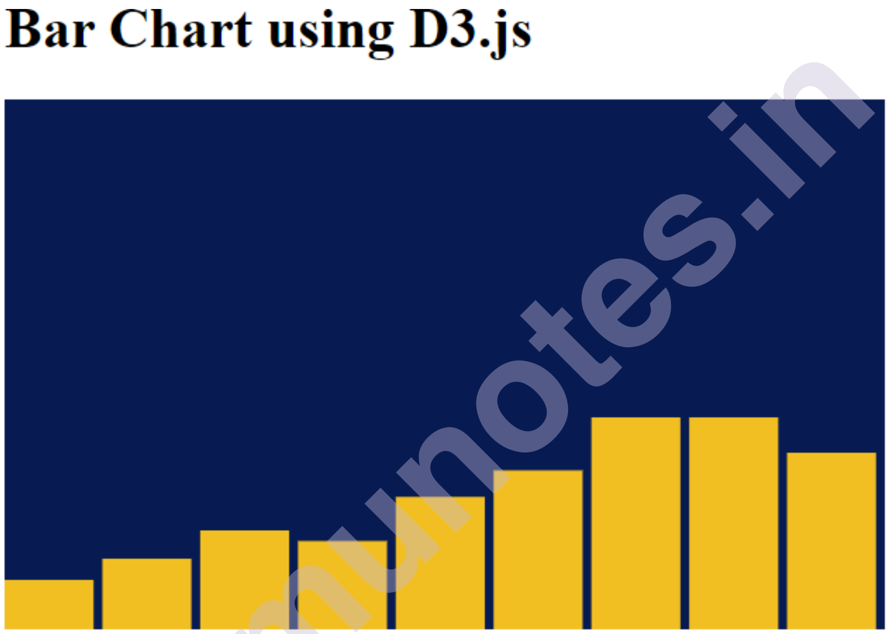
- Extremely flexible.
- Easy to use and fast.
- Supports large datasets.
- Declarative programming.
- Code reusability.
- Has wide variety of curve generating functions.
- Associates data to an element or group of elements in the html page.

D3.js Benefits:

D3.js is an open source project and works without any plugin. It requires very less code and comes up with the following benefits −

- Great data visualization.
- It is modular. You can download a small piece of D3.js, which you want to use. No need to load the whole library every time.
- Easy to build a charting component.
- DOM manipulation.

12.2 ANOTHER TWIST ON BAR CHART VISUALIZATION



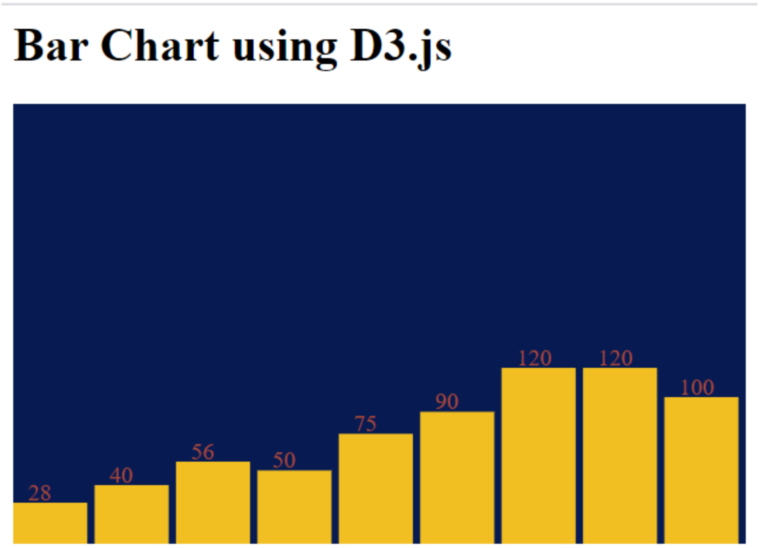
This shows the visualization but there is no precise information other than the trend. So our next task is to add some labels so that the values of each bar will be visible giving more information on visuals.

For that you can add following code snippet add the end of the script.js file.

script.js

```
var text =
svg.selectAll(“text”).data(dataset).enter().append(“text”).text(function(d)
{return d;}).attr(“y”, function(d, i) {return chartHeight — d —
2;}).attr(“x”, function(d, i) {return barWidth * i + 10;}).attr(“fill”,
“#A64C38”);
```

This will result something like below showing the values.

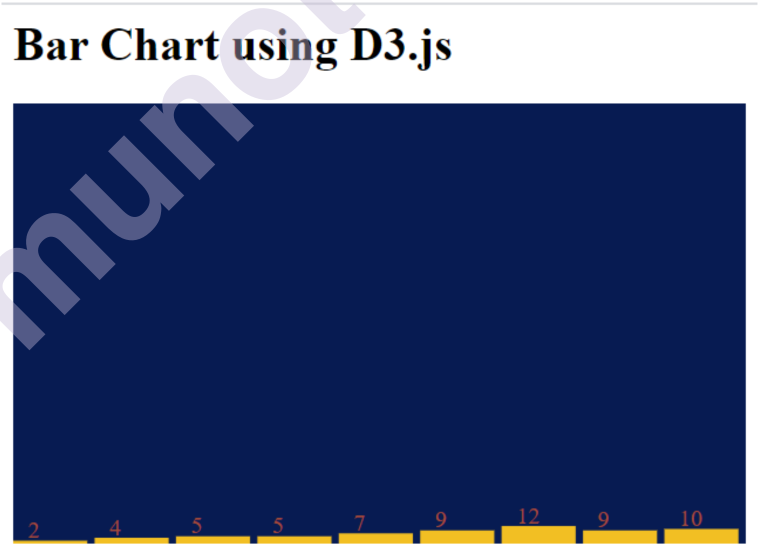


Bar Chart using D3.js

Now what are going to do is add scaling to our chart. You can have various values in your data set; some can be really small and some can be really large. So for better visualizations with consistency it's important to have scaling in your chart.

If we rearrange our dataset as below you can see how the bar chart is rendered.

```
var dataset = [2, 4, 5, 5, 7, 9, 12, 9, 10];
```



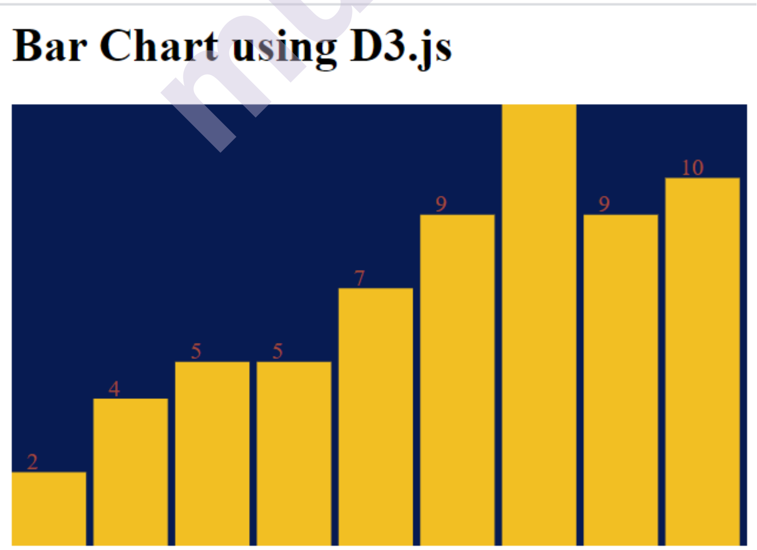
Bar Chart using D3.js

So this way we can barely see the bars in the chart. So we have to scale it up according to the chart height. If you have larger values in your dataset it will be not shown the accurate height within the given chart height. So the solution for this is scaling accordingly. For that we can use a scaler like this and change the bar chart.

script.js:

```
var dataset = [2, 4, 5, 5, 7, 9, 12, 9, 10];var chartWidth = 500, chartHeight
= 300, barPadding = 5;var barWidth = (chartWidth / dataset.length);var
```

```
svg = d3.select('svg')
.attr("width", chartWidth)
.attr("height", chartHeight);var yScale = d3.scaleLinear()
.domain([0, d3.max(dataset)])
.range([0, chartHeight])var barChart = svg.selectAll("rect")
.data(dataset)
.enter()
.append("rect")
.attr("y", function(d) {
return chartHeight — yScale(d);})
.attr("height", function(d) {
return yScale(d);})
.attr("width", barWidth — barPadding)
.attr("fill", '#F2BF23')
.attr("transform", function (d, i) {
var translate = [barWidth * i, 0];
return "translate("+ translate +")";
});var text = svg.selectAll("text")
.data(dataset)
.enter()
.append("text")
.text(function(d) {
return d;})
.attr("y", function(d, i) {
return chartHeight — yScale(d) — 2;})
.attr("x", function(d, i) {
return barWidth * i + 10;})
.attr("fill", "#A64C38");
```



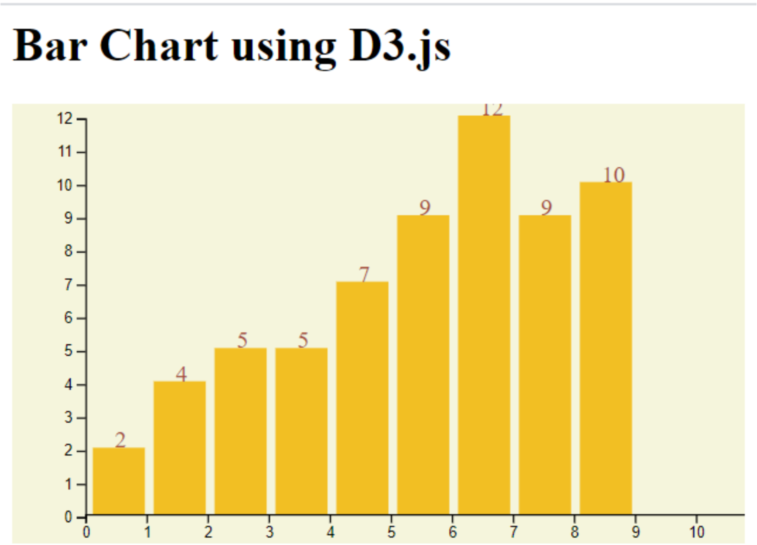
So now it's quite obvious that axes are missing out in our chart. So it's really simple to add axes to our graph using D3.js. You can create x-axis and y-axis using x-scale and y-scale in D3.js. You can create a scaled graph with labels using following code snippet.

script.js:

```
var dataset = [2, 4, 5, 5, 7, 9, 12, 9, 10];var chartWidth = 500, chartHeight
= 300, barPadding = 6;var barWidth = (chartWidth / dataset.length -
14);var svg = d3.select('svg')
.attr("width", chartWidth)
.attr("height", chartHeight);var xScale = d3.scaleLinear()
.domain([0, d3.max(dataset)])
.range([0, chartWidth]);var yScale = d3.scaleLinear()
.domain([0, d3.max(dataset)])
.range([0,chartHeight — 28]);var yScaleChart = d3.scaleLinear()
.domain([0, d3.max(dataset)])
.range([chartHeight — 28, 0]);var barChart = svg.selectAll("rect")
.data(dataset)
.enter()
.append("rect")
.attr("y", function(d) {
return chartHeight — yScale(d) — 20;})
.attr("height", function(d) {
return yScale(d);})
.attr("width", barWidth — barPadding)
.attr("fill", '#F2BF23')
.attr("transform", function (d, i) {var translate = [barWidth * i +55, 0];
return "translate("+ translate +")";});var text = svg.selectAll("text")
.data(dataset)
.enter()
.append("text")
.text(function(d) {
return d;})
.attr("y", function(d, i) {
return chartHeight — yScale(d) — 20;})
.attr("x", function(d, i) {
return barWidth * i + 70;})
.attr("fill", "#A64C38");var x_axis = d3.axisBottom().scale(xScale);var
y_axis = d3.axisLeft().scale(yScaleChart);svg.append("g")
.attr("transform", "translate(50, 10)")
.call(y_axis);var xAxisTranslate = chartHeight — 20;svg.append("g")
.attr("transform", "translate(50, " + xAxisTranslate +")")
.call(x_axis);
```

style.css

```
.bar-chart {background-color: beige;}
```

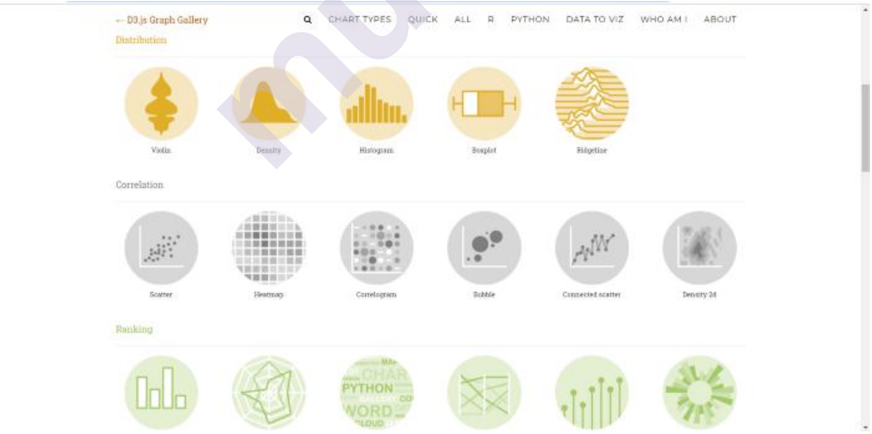


Bar Chart using D3.js

So like above simple bar chart we can create many types of graphs we want. The best thing here is the control we have over what we create. Unlike other ready-made graphs which has are limited-customizable, we can have the freedom of creating our own graphs using SVGs in D3.js.

D3 solves the cause of the problem: efficient manipulation of documents based on data. This avoids proprietary representation and affords extraordinary flexibility, exposing the full capabilities of web standards such as HTML, SVG, and CSS. With minimal overhead, D3 is extremely fast, supporting large datasets and dynamic behaviors for interaction and animation.

You can have a look at the already created complex and attractive graphs at D3.js graph gallery.



Whereas most people will refer to D3.js as a data visualization library, it's not. D3 is more of a framework comprising different parts such as jQuery parts (which help us select and manipulate DOM elements), Lodash parts, animation parts, data analysis parts, and data visualization parts.

*****

13

TABLEAU

Unit Structure

13.0 OBJECTIVE

Until the early 21st century, the Database were used to produce numbers and data. It's the job of IT professionals to analyse the data and create reports.

Tableau was founded by Pat Hanrahan, Christian Chabot, and Chris Stolte from Stanford University in 2003. The main idea behind its creation is to make the database industry interactive and comprehensive.

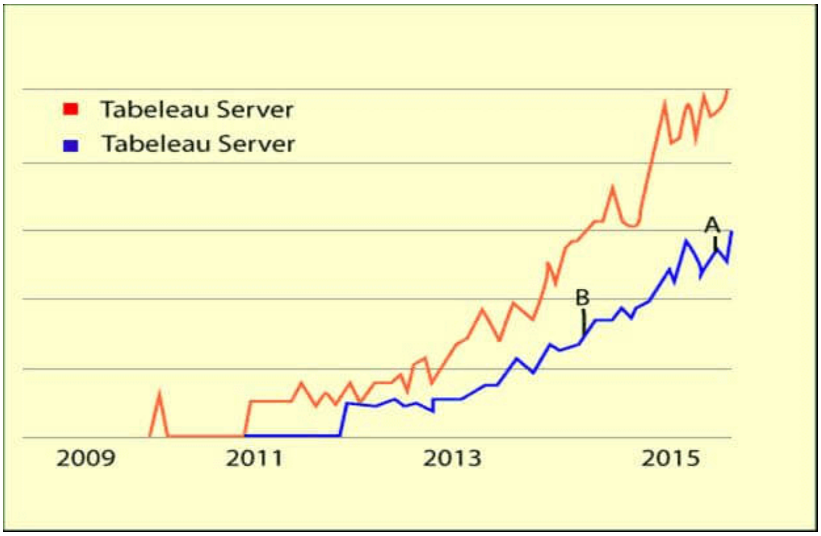
Tableau appears in the era when there were already established companies like Cognos, Microsoft Excel, Business Objects, etc. It managed to climb the success chart with $3.8 billion of current market value.

Since then, the company is growing day by day.

In August 2016, Tableau announced and appointed Adam Selipsky as president and CEO of the company.

What made Tableau Popular?:

The main logic behind creating this tool was developing a simple and user-friendly tool that can help you in creating graphs, charts, maps, reports as well as assist you in the next-gen concepts like the predictive and prescriptive analysis.



The worldwide business analytics market grew from $37.7 billion in 2013 to $59.2 billion in 2018, which translates to 9.4% compounded annual growth rate for the forecast period.

The main features that led Tableau Software to achieve success are:

- Powered by VizQL language, which makes it more flexible to pull data from any source.
- Provide Facility to the user with n number of visualization tools to customize the Tableau reports.
- All the complicated graphs and maps can be prepared with drags and drops method.
- Tableau data visualizations can be inserted with multiple platforms.
- It can analyze and display the data in real-time.

Some recently introduced versions of Tableau have the following features:

Tableau 9.0:

- ○ Smart maps
- ○ Instant visual feedback
- ○ Cashing and consolidation
- ○ Scalable and faster tableau server

Tableau 10.0:

- ○ Cluster analysis
- ○ Cross-database join
- ○ Self-service at scale
- ○ Multiple device support

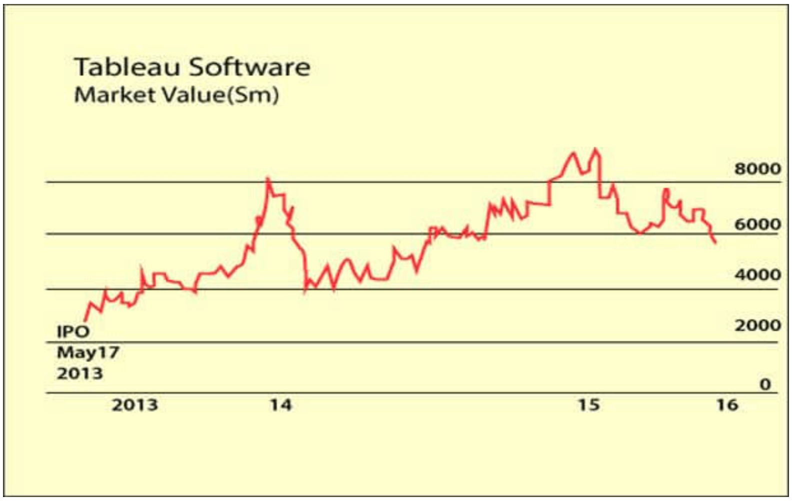


Tableau has seen a considerable growth of 82% in its annual sales over the past seven years from $18 million in 2009 to $654 million in 2015, making it to obtain the highest position in the ranking chart. This company now ranks under top 10 BI tools giving competition to other old tools like IBM, Microsoft, Qlik, Oracle, etc.

A report by Forbes in 2016 shows that the total income of Tableau grew 32% in the first quarter to $172 million, with foreign income up to 52%. The company closed 268 transactions greater than $100,000, up to 8% per year. If Tableau continues to perform with the same speed, its net worth will be in the $3 billion counted as one of the top three BI companies in the world.

13.1 INTRODUCTION

What is Tableau?

Tableau is the fastly growing and powerful data visualization tool. Tableau is a business intelligence tool which helps us to analyse the raw data in the form of the visual manner; it may be a graph, report, etc.

Example: If you have any data like **Big Data, Hadoop, SQL,** or any cloud data and if you want to analyse that given data in the form of pictorial representation of data, you can use Tableau.

Data analysis is very fast with Tableau, and the visualizations created are in the form of worksheets and dashboards. Any professional can understand the data created using Tableau.

Tableau software doesn't require any technical or any programming skills to operate. Tableau is easy and fast for creating visual dashboards.

Why use Tableau?

Here are some reasons to use Tableau:

- Ultimate skill for Data Science
- User-Friendly
- Apply to any Business

- Fast and Easy
- You don't need to do any Coding
- Community is Huge
- Hold the power of data
- It makes it easier to understand and explain the Data Reports

Features of Tableau:

- **Data Blending:** Data blending is the most important feature in Tableau. It is used when we combine related data from multiple data sources, which you want to analyze together in a single view, and represent in the form of a graph.

Example: Assume, we have Sales data in relational database and Sales Target data in an Excel sheet. Now, we have to compare actual sales with target sales, and blend the data based on common dimensions to get access. The two sources which are involved in data blending referred to as primary data and secondary data sources. A left join will be created between the primary data source and the secondary data source with all the data rows from primary and matching data rows from secondary data source to blend the data.

- **Real-time analysis:** Real-Time Analysis makes users able to quickly understand and analyse dynamic data, when the Velocity is high, and real-time analysis of data is complicated. Tableau can help extract valuable information from fast moving data with interactive analytics.
- **The Collaboration of data:** Data analysis is not isolating task. That's why Tableau is built for collaboration. Team members can share data, make follow up queries, and forward easy-to-digest visualizations to others who could gain value from the data. Making sure everyone understands the data and can make informed decisions is critical to success.

13.2 TABLEAU AS A VISUALIZATION TOOL

Top 10 Data Visualization Tools

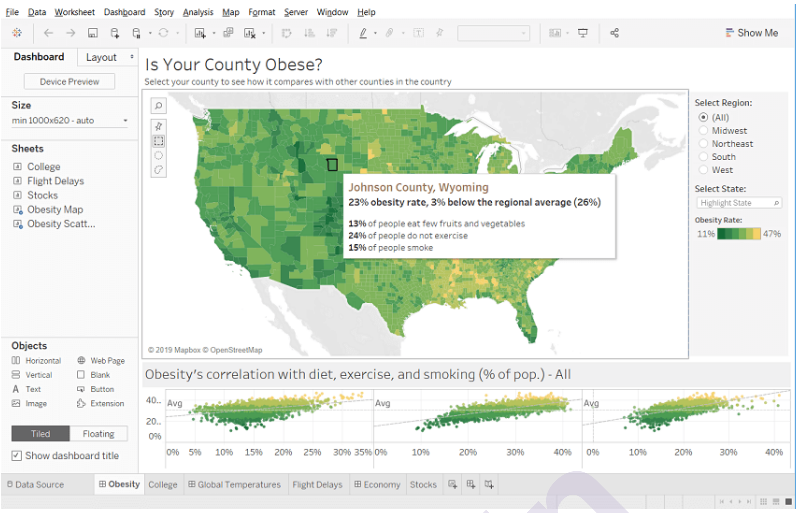
There are tools which help you to visualize all your data in a few minutes. They are already there; only you need to do is to pick the right data visualization tool as per your requirements.

Data visualization allows you to interact with data. **Google**, **Apple**, **Facebook**, and **Twitter** all ask better a better question of their data and make a better business decision by using data visualization.

Here are the top 10 data visualization tools that help you to visualize the data:

1. Tableau:

Tableau is a data visualization tool. You can create graphs, charts, maps, and many other graphics.



Is Your County Obese?

Select your county to see how it compares with other counties in the country

Johnson County, Wyoming

23% obesity rate, 3% below the regional average (26%)

13% of people eat few fruits and vegetables

24% of people do not exercise

15% of people smoke

Obesity's correlation with diet, exercise, and smoking (% of pop.) - All

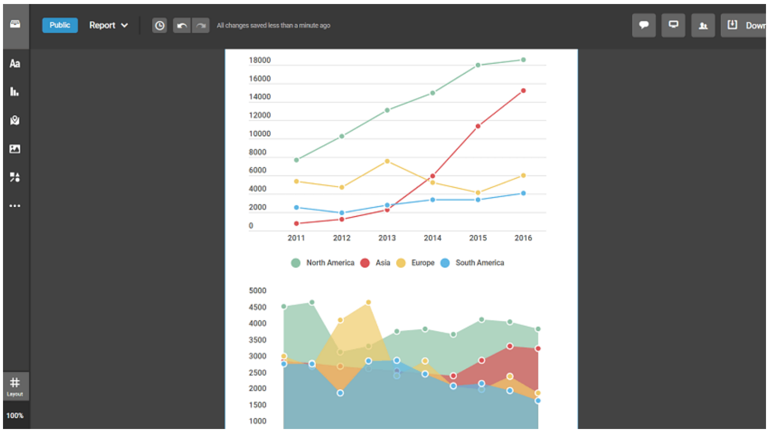
A tableau desktop app is available for visual analytics. If you don't want to install tableau software on your desktop, then a server solution allows you to visualize your reports online and on mobile.

A cloud-hosted service also is an option for those who want the server solution but don't want to set up manually. The customers of Tableau include Barclays, Pandora, and Citrix.

2. Infogram:

Infogram is also a data visualization tool. It has some simple steps to process that:

1. First, you choose among many templates, personalize them with additional visualizations like maps, charts, videos, and images.
2. Then you are ready to share your visualization.
3. Infogram supports team accounts for journalists and media publishers, branded designs of classroom accounts for educational projects, companies, and enterprises.



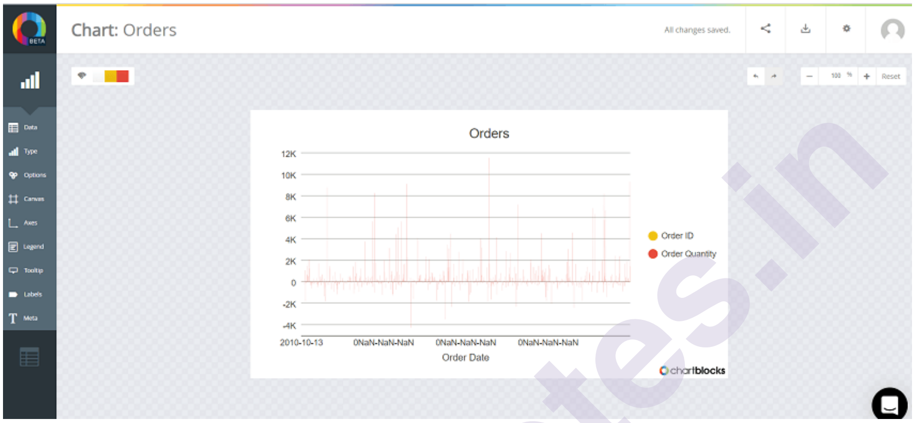
All changes saved less than a minute ago

An infogram is a representation of information in a graphic format designed to make the data easily understandable in a view. Infogram is used to quickly communicate a message, to simplify the presentation of large amounts of the dataset, to see data patterns and relationships, and to monitor changes in variables over time.

Infogram abounds in almost any public environment such as traffic signs, subway maps, tag clouds, musical scores, and weather charts, among a huge number of possibilities.

3. Chartblocks:

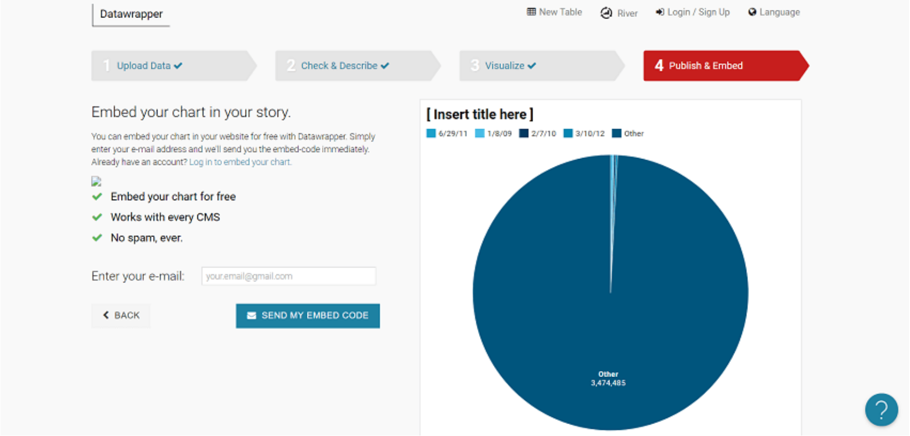
Chart blocks is an easy way to use online tool which required no coding and builds visualization from databases, spreadsheets, and live feeds.



Your chart is created under the hood in **html5** by using the powerful JavaScript library **D3.js**. Your visualizations is responsive and compatible with any screen size and device. Also, you will be able to embed your charts on any web page, and you can share it on Facebook and Twitter.

4. Datawrapper:

Datawrapper is an aimed squarely at publisher and journalist. **The Washington Post, VOX, The Guardian, BuzzFeed, The Wall Street Journal** and **Twitter** adopts it.



Embed your chart in your story.

You can embed your chart in your website for free with Datawrapper. Simply enter your e-mail address and we'll send you the embed-code immediately. Already have an account? Log in to embed your chart.

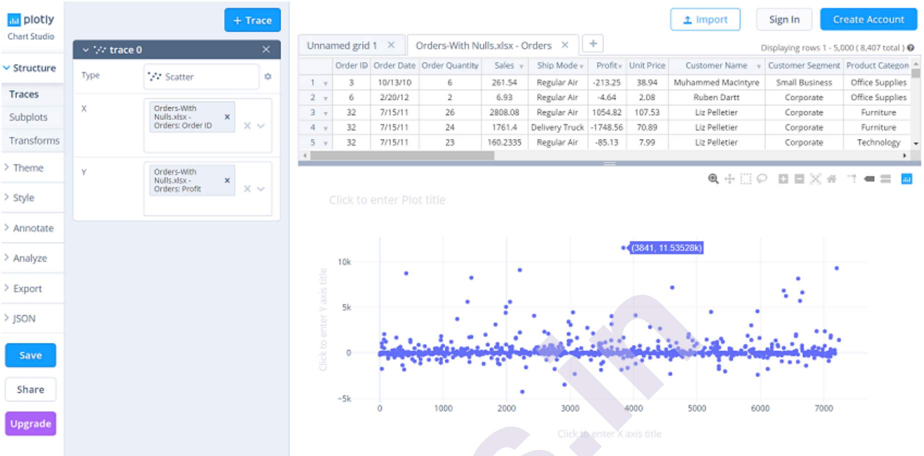
✓ Embed your chart for free
✓ Works with every CMS
✓ No spam, ever.

Enter your e-mail: your.email@gmail.com

Datawrapper is easy visualization tool, and it requires zero coding's. You can upload your data and easily create and publish a map or a chart. The custom layouts to integrate your visualizations perfectly on your site and access to local area maps are also available.

5. Plotly:

Plotly will help you to create a slick and sharp chart in just a few minutes or in a very short time. It also starts from a simple spreadsheet.



Type: Scatter

X: Orders-With Nulls.xlsx - Orders: Order ID

Y: Orders-With Nulls.xlsx - Orders: Profit

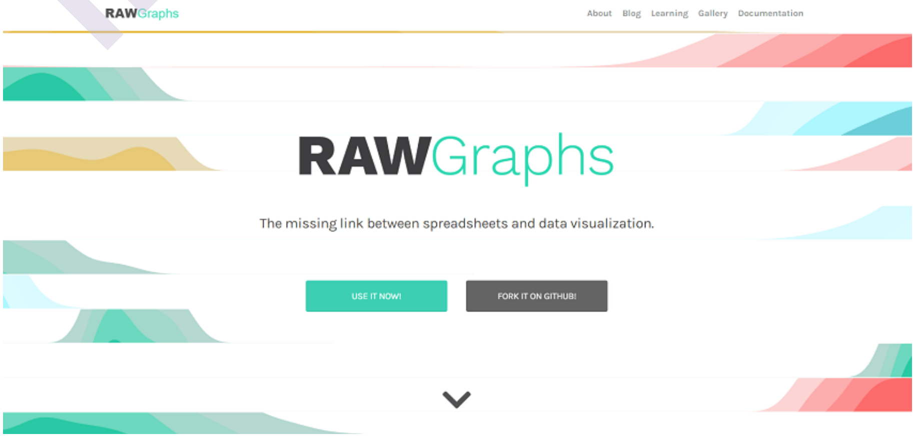
	Order ID	Order Date	Order Quantity	Sales	Ship Mode	Profit	Unit Price	Customer Name	Customer Segment	Product Categon
1	3	10/13/10	6	261.54	Regular Air	-213.25	38.94	Muhammed MacIntyre	Small Business	Office Supplies
2	6	2/20/12	2	6.93	Regular Air	-4.64	2.08	Ruben Dartt	Corporate	Office Supplies
3	32	7/15/11	26	2808.08	Regular Air	1054.82	107.53	Liz Pelletier	Corporate	Furniture
4	32	7/15/11	24	1761.4	Delivery Truck	-1748.56	70.89	Liz Pelletier	Corporate	Furniture
5	32	7/15/11	23	160.2335	Regular Air	-85.13	7.99	Liz Pelletier	Corporate	Technology

The guys use Plotly at Google and also by the **US Air Force, Goji** and **The New York University**.

Plotly is very user-friendly visualization tool which is quickly started within a few minutes. If you are a part of a team of developers that wants to have a crack, an API is available for JavaScript and Python languages.

6. RAW:

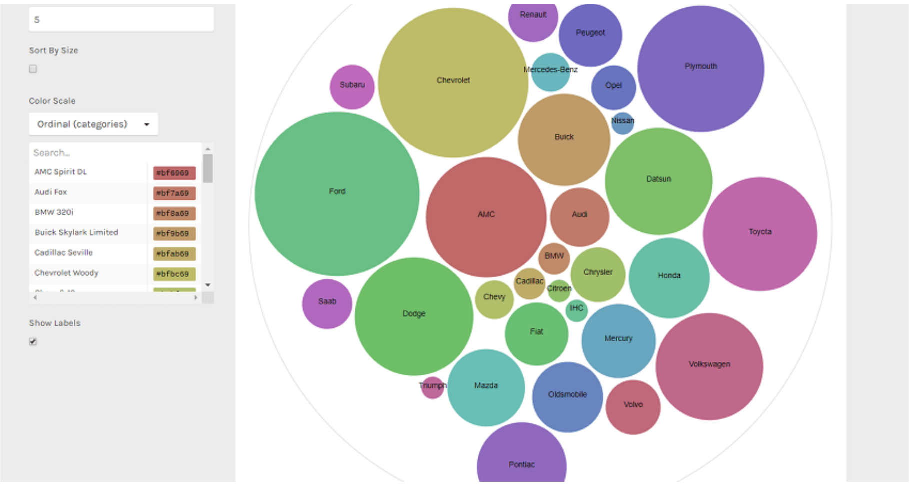
RAW creates the missing link between spreadsheets and vector graphics on its home page.



The missing link between spreadsheets and data visualization.

Your Data can come from **Google Docs, Microsoft Excel, Apple Numbers**, or a simple comma-separated list.

Here the kicker is that you can export your visualization easily and have a designer to make it look sharp. RAW is compatible with Inkscape, Adobe Illustrator, and Sketch. RAW is very easy to use and get quick results.



5

Sort By Size

□

Color Scale

Ordinal (categories)

Search...

AMC Spirit DL	#bf6969
Audi Fox	#bf7a69
BMW 320i	#bf8a69
Buick Skylark Limited	#bf9b69
Cadillac Seville	#bfab69
Chevrolet Woody	#bfbc69

Show Labels

☑

7. Visual.ly:

Visual.ly is a visual content service. It has a dedicated data visualization service and their impressive portfolio that includes work for **Nike, VISA, Twitter, Ford, The Huffington post**, and the national geographic.

Visual Content for Modern Marketers

A new way to think about content creation

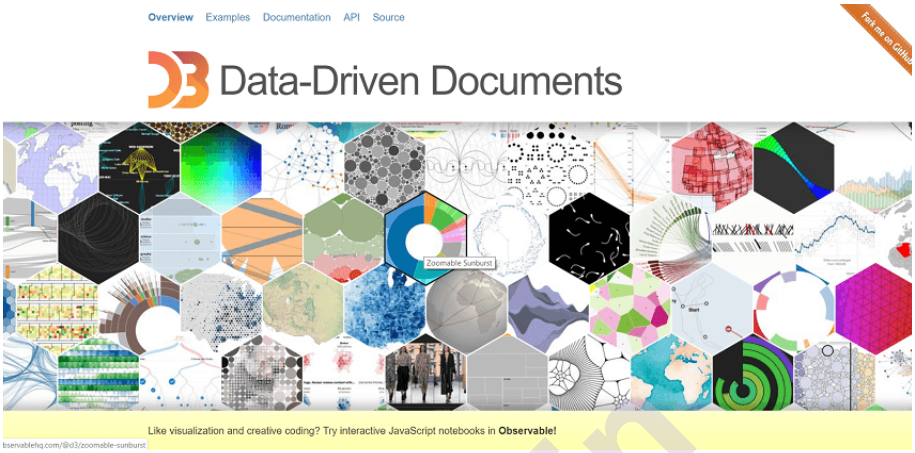
Specialized creative talent, online and on-demand

By a streamlined online process, you can find entire outsource your visualizations to a third-party where you describe your project and connected with a creative team that will stay with you for the entire duration of the project.

Visual.ly sends you an email notification for all the event you are hitting, and also it will give you constant feedback to your creative team. Visual.ly offer their distribution network for showcasing your project after it's completed.

8. D3.js:

D3.js is a best data visualization library for manipulating documents. D3.js runs on JavaScript, and it uses **CSS**, **html**, and **SVG**. D3.js is an open-source and applies a data-driven transformation to a webpage. It's only applied when data is in **JSON** and **XML** file.



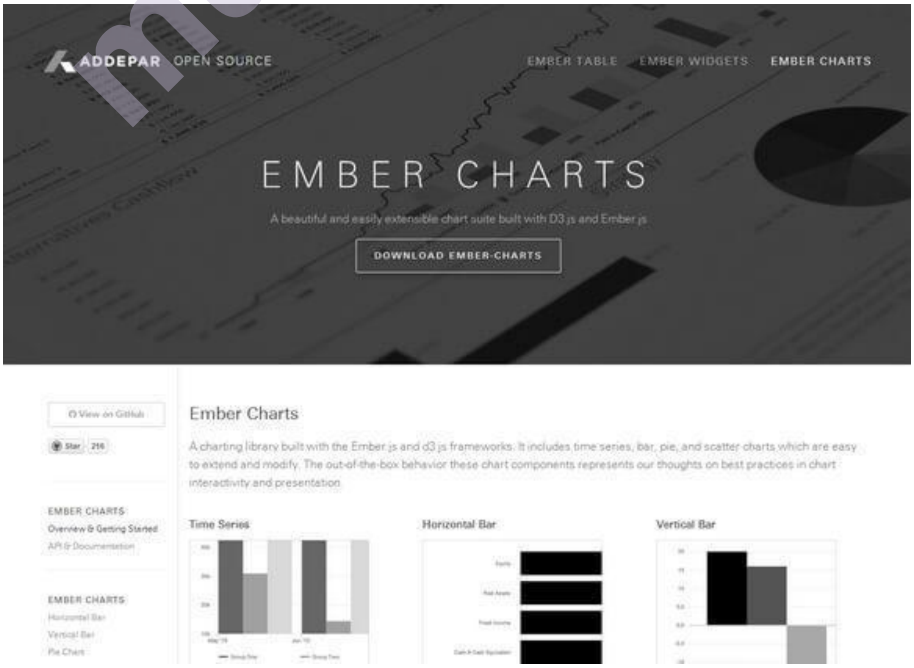
Like visualization and creative coding? Try interactive JavaScript notebooks in **Observable!**

D3.js emphasis on web standards gives you the full capabilities of modern browsers without tying yourself to a single framework and combining powerful visualization components.

D3.js is as powerful as it is a cutting-edge library, so it comes with no pre-built charts and only IE9+ supports this library.

9. Ember Charts:

Ember charts are based on the ember.js and D3.js framework, and it uses the D3.js under the hood. It also applied when the data is in **JSON** and **XML** file.



A beautiful and easily extensible chart suite built with D3.js and Ember.js

Ember Charts

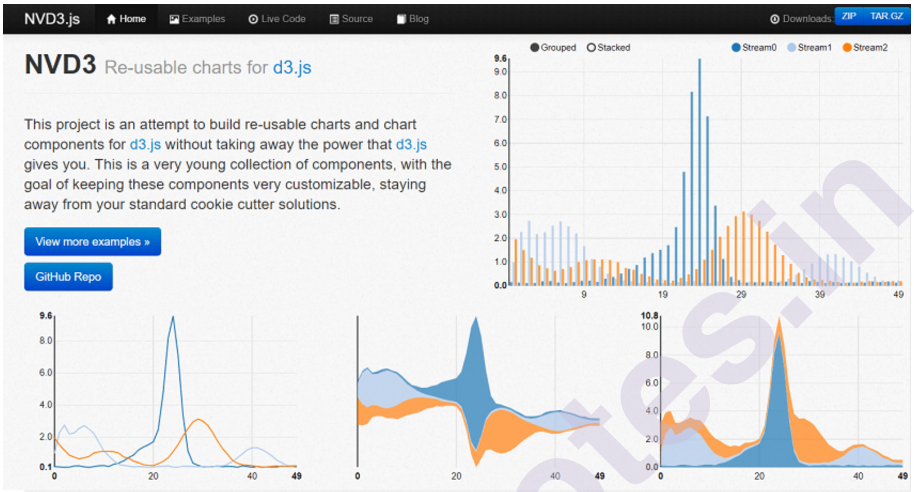
A charting library built with the Ember.js and d3.js frameworks. It includes time series, bar, pie, and scatter charts which are easy to extend and modify. The out-of-the-box behavior these chart components represents our thoughts on best practices in chart interactivity and presentation.

It includes a bar, time series, pie, and scatter charts which are easy to extend and modify. These chart components represent our thoughts on best practices in chart presentation and interactivity.

The team behind Ember Charts is also the same that created Ember.js. It puts a lot of focus on best practices and interactivity. Error handling is very graceful, and your app will not crash after finding irrelevant data or corrupt data.

10. NVD3:

NVD3 is a project that attempts to build reusable charts and components. This project is to keeps all your charts neat and customizable.



NVD3 Re-usable charts for d3.js

This project is an attempt to build re-usable charts and chart components for d3.js without taking away the power that d3.js gives you. This is a very young collection of components, with the goal of keeping these components very customizable, staying away from your standard cookie cutter solutions.

NDV3 is a simpler interface on the top of the D3.js and keeps all of its powerful features under the hood.

The front end engineers develop NDV3, and they use their insight into charting technology. This charting technology is used to provide powerful analytics to clients in the financial industry.

13.3 DASHBOARD- FOR BIG DATA VISUALIZATION

What is a data visualization dashboard?

A dashboard is a data visualization tool that tracks, analyzes, and displays KPIs, metrics, and critical data points. Dashboards empower both technical and non-technical users to understand and leverage business intelligence to make more informed decisions. Users actively participate in the analytics process by compiling data and visualizing trends or occurrences, and uncovering an objective view of performance metrics that can be immediately understood.

Dashboards feature visualized data via charts, tables, and gauges. Viewers use these visualizations to monitor the health of the organization against established goals and industry benchmarks.

The benefits of using dashboards:

Visualize multiple KPIs at once:

Most organizations utilize a variety of services to track KPIs and metrics, including marketing automation platforms, email marketing platforms, CRM tools, and many more. Monitoring and analysing the data from each of these tools individually wastes precious time and resources.

Using a data visualization dashboard, users receive a bird's eye view of the data from each of these platforms in one centralized location, with the ability to quickly understand what it means for the business. The user can then drill down deeper into any aspect of the data, comparing it to established KPIs, which help us to understand what's working and where there's room for improvement.

Make data easier to understand:

You don't need to be a data scientist to use and understand a dashboard. The average user can quickly scan data visualization dashboards, obtaining a high-level view of key data points without painstakingly sorting through spreadsheets, emails, or documents for the answers to critical business questions.

Increase accessibility and collaboration:

Dashboards make it easy for teams to collaborate whether everyone works in the office, virtually, or out in the field. Cloud-based dashboard tools update in real time and are accessible from any browsers. They help keep everyone on the same page and working toward the same goals.

Users can also create a link to their dashboards that can be shared with stakeholders inside and outside of the organization.

Create reports on the fly

In today's fast-paced global business environment, it's critical to ditch the old habit of generating reports at the end of the month, quarter, or year. By utilizing dashboards that update in real time, your organization can make swift changes before they have a chance to cause significant harm to the business.

Leadership doesn't have to wait on management to provide them with reports; everyone in the organization can easily access a dashboard and use its insights for intelligent decision making.

How to build a data visualization dashboard:

The best data visualization dashboards are designed by first asking a critical business question. Let's take a challenge such as tracking sales revenue within an organization. For example, an executive may want to know "How much revenue did the business generate this quarter? Was it more or less than the last quarter? And what regions performed the highest?"

Extract your data:

Import data into your dashboard from your data warehouse, via web services like Google Analytics, or by uploading old-fashioned Excel or CSV files. If you're using a modern data visualization tool, take advantage of its integrations that allow you to extract data from 100s of other cloud platforms your company may use.

Once data is collected in your preferred analytics tool, you'll want to take that data set and apply and begin to build visualizations to answer your questions.

Determine your KPIs:

What metrics will you use your data visualization dashboard to track? Remember, your dashboard should provide an answer to a specific business question, so keep this in mind when determining the KPIs unique to your department, line of business, company, or industry.

When designing your dashboard, keep your audience top of mind. While an executive may want to see a high-level overview of performance across every marketing channel, a marketing manager is more interested in the day to day performance, but the ROI of your marketing investment may be an essential metric for the entire department to know.

Common types of dashboards:

It’s critical to cultivate a data-driven culture in today’s competitive business landscape. That begins with making data available to team’s within an organization. Dashboards are a great place to start, they help raise deeper questions and encourage the natural curiosity of employees.

Here’s some examples of common dashboards and how they are used across business units.

Marketing:

Use dashboards to track the performance of your digital marketing programs—including social media engagement, SEO, web traffic, and email open rates—or better understand how your marketing funnel is performing by tracking leads generated across various channels.

Sales:

Use a dashboard to understand the performance of individual salespeople, or track closed or lost deals, pipeline conversion rates, scheduled demos and more to help the sales team understand how they are performing and where they can best direct their efforts to hit company targets.

Engineering:

Software engineers rely on dashboards to track product development, usage, and performance. Dashboards make it easy to communicate important metrics and spot issues before they become large scale problems.

Executive:

Decision-makers typically use dashboards to track company goals, catch trends early, inform strategy, and understand whether business efforts are paying off. Data visualization software provides a real-time overview of business processes, help to control the workflow of a company, and use dashboards to visualize multiple KPIs in one place.

Real-time data visualization dashboards help you to concentrate on important KPIs via interactive charts, graphs, and tables that represent the current state of your company and empower you to make data-driven decisions that provide a competitive edge.

*****