

**NM1059 - BIG DATA BY INFOSYS**

**A PROJECT REPORT**

***Submitted by***

**DEPARTMENT**

**OF**

**INFORMATION TECHNOLOGY**

**Dr. SIVANTHI ADITANAR COLLEGE OF ENGINEERING**

**TIRUCHENDUR-628215**

**ANNA UNIVERSITY : CHENNAI**  **600025**

**DECEMBER 2024**

**ANNA UNIVERSITY : CHENNAI 600 025**

BONAFIDE CERTIFICATE

Certified that this Naan Mudhalvan project report “**BIG DATA**” is the bonafide work of who carried out the project at “**Infosys”.**

## SIGNATURE

Mrs.K.P.Ramya, M.Tech(PhD)

# STAFF INCHARGE

## ASSISTANT PROFESSOR

Department of Information Technology

Dr. Sivanthi Aditanar College Engineering,

Tiruchendur- 628215

## SIGNATURE

Dr.S.SeIvi, M.E, Ph.D., MIE, MISTE

# HEAD OF THE DEPARTMENT

## PROFESSOR

Department of Information Technology

Dr. Sivanthi Aditanar College of Engineering,

Tiruchendur-628215

**INTERNAL EXAMINER EXTERNAL EXAMINER**

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| **CHAPTER**  **NO** | **TITLE** | **PAGE**  **NO** |
| **1** | **INTRODUCTION** | **1** |
|  | **1.1 BIG DATA** | **1** |
|  | **1.2 HADOOP** | **2** |
|  | **1.3 HIVE** | **3** |
|  | **1.4 SCALA PROGRAMMING** | **4** |
|  | **1.5 SPARK** | **5** |
| **2** | **SYSTEM SPECIFICATIONS** | **7** |
|  | **2.1 SOFTWARE REQUIREMENTS** | **7** |
|  | **2.2 HARDWARE REQUIREMENTS** | **8** |
| **3** | **IMPLEMENTATION** | **9** |
|  | **3.1 PROBLEM STATEMENT** | **9** |
|  | **3.2 INSTRUCTIONS FOR PROBLEM SOLVING** | **9** |
|  | **3.3 HADOOP INSTALLATION** | **10** |
|  | **3.4 HIVE INSTALLATION** | **16** |
|  | **3.5 SPARK INSATLLATION AND TASKS** | **18** |
| **4** | **CONCLUSION** | **31** |
| **5** | **CERTIFICATE** | **33** |

**CHAPTER 1**

**INTRODUCTION**

* 1. **BIG DATA**

Big data technology is defined as software-utility. This technology is primarily designed to analyze, process and extract information from a large data set and a huge set of extremely complex structures. This is very difficult for traditional data processing software to deal with.

Among the larger concepts of rage in technology, big data technologies are widely associated with many other technologies such as deep learning, machine learning, artificial intelligence (AI), and Internet of Things (IoT) that are massively augmented. In combination with these technologies, big data technologies are focused on analyzing and handling large amounts of real-time data and batch-related data.

Big Data is typically managed and analyzed using advanced tools and frameworks such as:

* **Hadoop** and **Spark** for distributed data storage and processing.
* **NoSQL databases** like MongoDB and Cassandra for flexible data handling.
* **Machine learning** and **AI models** to extract meaningful patterns and predictions

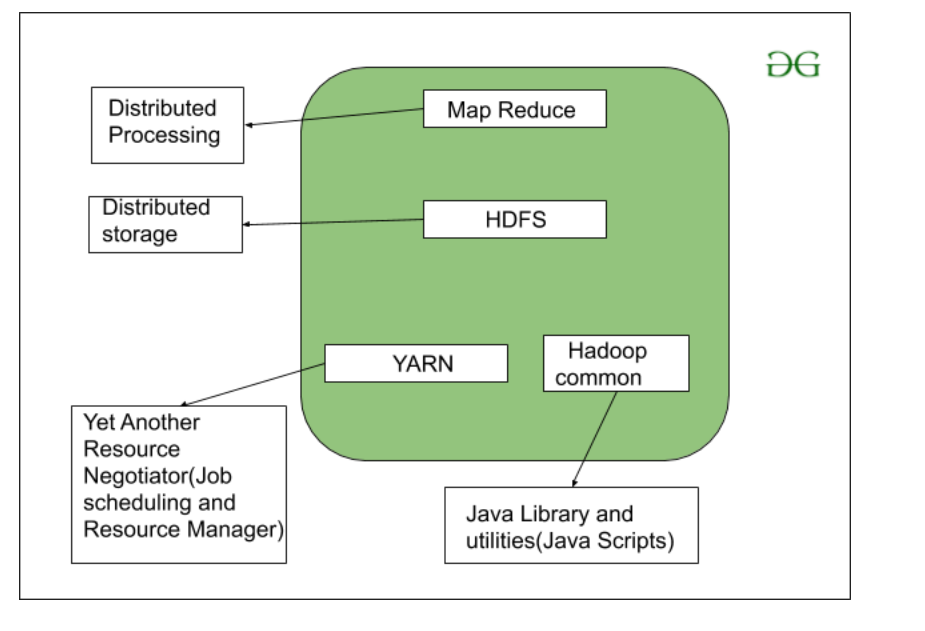
**Key Characteristics of Big Data (The 5 Vs):**

1. **Volume**: The sheer amount of data, ranging from terabytes to petabytes and beyond.
2. **Velocity**: The speed at which data is generated, collected, and processed, often in real-time.
3. **Variety**: The diverse formats and types of data, including text, images, audio, video, and log files.
4. **Veracity**: The uncertainty and reliability of data, highlighting the need for accurate and trustworthy sources.
5. **Value**: The insights and business advantages derived from analyzing big data.
   1. **HADOOP**

Hadoop is an open source software programming framework for storing a large amount of data and performing the computation. Its framework is based on Java programming with some native code in C and shell scripts. It is designed to handle big data and is based on the MapReduce programming model, which allows for the parallel processing of large datasets.

* HDFS (Hadoop Distributed File System): This is the storage component of Hadoop, which allows for the storage of large amounts of data across multiple machines. It is designed to work with commodity hardware, which makes it cost-effective.
* YARN (Yet Another Resource Negotiator): This is the resource management component of Hadoop, which manages the allocation of resources (such as CPU and memory) for processing the data stored in HDFS.
* Hadoop also includes several additional modules that provide additional functionality, such as Hive (a SQL-like query language), Pig (a high-level platform for creating MapReduce programs), and HBase (a non-relational, distributed database).
* Hadoop is commonly used in big data scenarios such as data warehousing, business intelligence, and machine learning. It’s also used for data processing, data analysis, and data mining. It enables the distributed processing of large data sets across clusters of computers using a simple programming model.

**ARCHITECTURE**



* 1. **HIVE**

Hive is a data warehouse system which is used to analyze structured data. It is built on the top of Hadoop. It was developed by Facebook.

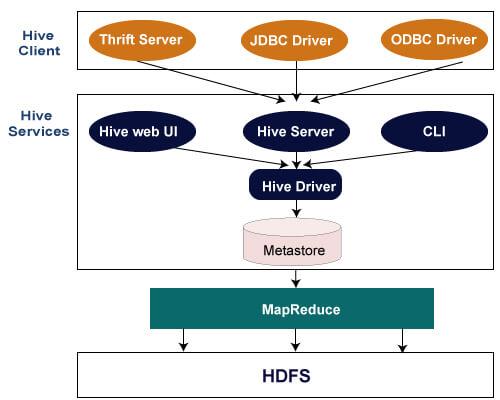
Hive provides the functionality of reading, writing, and managing large datasets residing in distributed storage. It runs SQL like queries called HQL (Hive query language) which gets internally converted to MapReduce jobs.

Using Hive, skip the requirement of the traditional approach of writing complex MapReduce programs can be skipped. Hive supports Data Definition Language (DDL), Data Manipulation Language (DML), and User Defined Functions (UDF).

**FEATURES**

* Hive is fast and scalable.
* It provides SQL-like queries (i.e., HQL) that are implicitly transformed to MapReduce or Spark jobs.
* It is capable of analyzing large datasets stored in HDFS.
* It allows different storage types such as plain text, RCFile, and HBase.
* It uses indexing to accelerate queries.
* It can operate on compressed data stored in the Hadoop ecosystem.
* It supports user-defined functions (UDFs) where user can provide its functionality.

**ARCHITECTURE**

****

* 1. **SCALA**

Scala is a general-purpose, high-level, multi-paradigm programming language. It is a pure object-oriented programming language which also provides support to the functional programming approach. Scala programs can convert to bytecodes and can run on the JVM (Java Virtual Machine). Scala stands for Scalable language. It also provides Javascript runtimes. Scala is highly influenced by Java and some other programming languages like Lisp, Haskell, Pizza etc.

**Scala's Role in Big Data Frameworks**

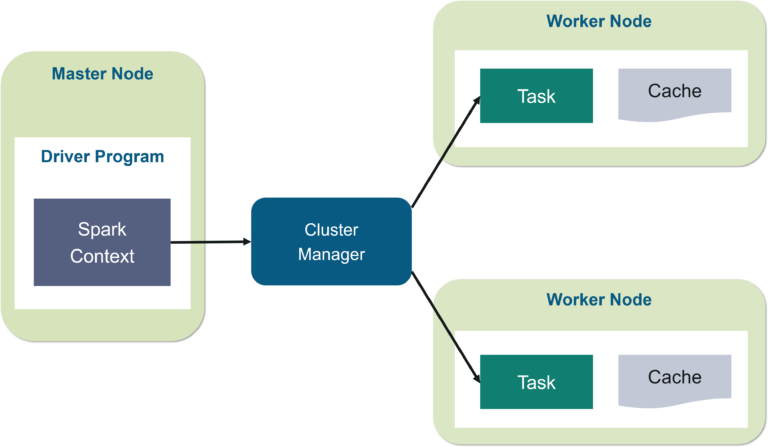
1. **Apache Spark**:
   * Spark is implemented in Scala, and its APIs for data manipulation and analytics are natively supported in Scala.
   * Features like RDDs (Resilient Distributed Datasets), DataFrames, and Datasets are optimized for Scala.
   * Scala's functional constructs, like map, reduce, and filter, align well with Spark's transformations and actions.
2. **Kafka Streams**:
   * Kafka, a distributed event-streaming platform, provides Scala APIs for building robust stream-processing applications.
3. **Akka**:
   * Akka is a toolkit for building concurrent, distributed systems in Scala, often used in real-time big data applications.
4. **Big Data Pipelines**:
   * Scala is often used in data engineering workflows for ETL (Extract, Transform, Load) operations in distributed systems.
   1. **SPARK**

Apache Spark is a powerful, open-source unified analytics engine designed for processing and analyzing large datasets. It provides high-speed computation and supports a wide range of big data operations, making it one of the most popular frameworks in the big data ecosystem.

**Core Components of Apache Spark**

1. **Spark Core**:
   * The foundational engine responsible for scheduling, memory management, fault recovery, and interacting with storage systems.
   * Implements Resilient Distributed Datasets (RDDs), the fundamental abstraction for distributed data.
2. **Spark SQL**:
   * Enables querying of structured and semi-structured data using SQL and DataFrames.
   * Supports integration with Hive for advanced data warehousing.
3. **Spark Streaming**:
   * Provides real-time stream processing capabilities for data from sources like Kafka, Flume, or socket streams.
4. **MLlib (Machine Learning Library)**:
   * Offers distributed algorithms for classification, regression, clustering, and recommendation.
5. **GraphX**:
   * A library for graph processing and analytics, such as page ranking and community detection.
6. **Spark Structured Streaming**:
   * A newer API for real-time data processing with better fault tolerance and scalability compared to Spark Streaming.

**ARCHITECTURE**

****

**CHAPTER 2**

**SYSTEM SPECIFICATIONS**

**2.1 SOFTWARE REQUIREMENTS**

* MySQL
* Hadoop 2.8.0
* Hive 2.3.0
* Sqoop 1.4.6
* Spark 2.x
* JDK 1.8 or above
* Eclipse IDE

**SOFTWARE DESCRIPTION**

* **MySQL:** MySQL is an open-source, relational database management system (RDBMS) developed by MySQL AB and now owned by Oracle Corporation. It follows the client-server model and is widely used in web development, business applications, and data warehousing.
* **Hadoop 2.8.0:** Hadoop 2.8.0 is a version of the Apache Hadoop project, released in March 2017, as part of the Hadoop 2.x series. This version introduced various improvements and fixes over previous releases, enhancing the stability, performance, and functionality of the Hadoop ecosystem.
* **Hive 2.3.0:** Apache Hive 2.3.0, released in July 2017, introduced several improvements, optimizations, and new features aimed at enhancing performance, usability, and integration within the Hadoop ecosystem. It continued Hive's role as a key data warehouse system for querying and managing large datasets stored in distributed systems like HDFS.
* **Sqoop 1.4.6:** Apache Sqoop 1.4.6, released in August 2015, is a tool designed to transfer data between Hadoop and structured data stores, such as relational databases and enterprise data warehouses. Sqoop simplifies the process of importing data from external systems into Hadoop Distributed File System (HDFS), as well as exporting data from Hadoop to relational databases.
* **Spark 2.x:** Apache Spark 2.x is a major version of the open-source distributed computing system, released to provide faster processing, enhanced performance, and more robust APIs.
* **JDK 1.8:** Also known as Java 8, was released by Oracle in March 2014. It is one of the most significant updates to the Java programming language, introducing a wide range of features that enhance the language’s expressiveness, performance, and ease of use.
* **Eclipse IDE:** Highly popular, open-source integrated development environment (IDE) primarily used for Java development but also supports various other programming languages through plugins.

**2.2 HARDWARE REQUIREMENTS**

* i5 or i7 processor or R5 from AMD
* 16 GB of RAM,500 GB storage system

**CHAPTER 3**

**IMPLEMENTATION**

**3.1 PROBLEM STATEMENT**

The pandemic Covid has badly impacted everyone's life across the globe in the year 2020. Assessing the available data related to patients, treatments, post Covid prognosis, recovery rate and many other such information will help hospitals and health organizations to evaluate what care approaches are most effective. This can also help in understanding what is the effect of medication on patients with history of other illness such as cardiac problems, diabetics, cancer etc.

All data related to Covid pandemic have continuously been monitored and analyzed to find the intensity of its spread. A sample of such data that has been captured from different locations on daily basis. You are required to get some useful insights by processing this data using the Big Data platform Hadoop and its ecosystem components.

Listed below are few reports expected from analysis:

* What is the number of people who are infected globally?
* How many cases are reported in a continent?
* Which country has recorded maximum number of deaths due to Covid?
* How many people are vaccinated so far?

**3.2 INSTRUCTION FOR PROBLEM SOLVING**

**Datasource:** CovidGlobalData.csv

The file contains the details about Covid infections worldwide.File structure is given as below.

|  |  |
| --- | --- |
| iso\_code | String |
| continent | String |
| Location | String |
| Date\_current | String |
| Total cases | Double |
| Total\_deaths | Double |
| Total\_vaccinations | Double |
| People\_vaccinated | Double |
| Median\_age | Double |
| Age\_65\_older | Double |
| Age\_70\_older | Double |
| Cardiovasc\_death\_rate | Double |
| Diabetes\_prevalence | Double |

**3.3 HADOOP INSTALLATION**

Prerequisite Test

=============================

sudo apt update

sudo apt install openjdk-8-jdk -y

java -version; javac -version

sudo apt install openssh-server openssh-client -y

sudo adduser hdoop

su - hdoop

ssh-keygen -t rsa -P '' -f ~/.ssh/id\_rsa

cat ~/.ssh/id\_rsa.pub >> ~/.ssh/authorized\_keys

chmod 0600 ~/.ssh/authorized\_keys

ssh localhost

Downloading Hadoop (Please note link is updated to new version of hadoop here on 6th May 2022)

===============================

wget https://downloads.apache.org/hadoop/common/hadoop-3.2.3/hadoop-3.2.3.tar.gz

tar xzf hadoop-3.2.3.tar.gz

Editng 6 important files

=================================

1st file

===========================

sudo nano .bashrc - here you might face issue saying hdoop is not sudo user

if this issue comes then

su - aman

sudo adduser hdoop sudo

sudo nano .bashrc

#Add below lines in this file

#Hadoop Related Options

export HADOOP\_HOME=/home/hdoop/hadoop-3.2.3

export HADOOP\_INSTALL=$HADOOP\_HOME

export HADOOP\_MAPRED\_HOME=$HADOOP\_HOME

export HADOOP\_COMMON\_HOME=$HADOOP\_HOME

export HADOOP\_HDFS\_HOME=$HADOOP\_HOME

export YARN\_HOME=$HADOOP\_HOME

export HADOOP\_COMMON\_LIB\_NATIVE\_DIR=$HADOOP\_HOME/lib/native

export PATH=$PATH:$HADOOP\_HOME/sbin:$HADOOP\_HOME/bin

export HADOOP\_OPTS"-Djava.library.path=$HADOOP\_HOME/lib/nativ"

source ~/.bashrc

2nd File

============================

sudo nano $HADOOP\_HOME/etc/hadoop/hadoop-env.sh

#Add below line in this file in the end

export JAVA\_HOME=/usr/lib/jvm/java-8-openjdk-amd64

3rd File

===============================

sudo nano $HADOOP\_HOME/etc/hadoop/core-site.xml

#Add below lines in this file(between "<configuration>" and "<"/configuration>")

<property>

<name>hadoop.tmp.dir</name>

<value>/home/hdoop/tmpdata</value>

<description>A base for other temporary directories.</description>

</property>

<property>

<name>fs.default.name</name>

<value>hdfs://localhost:9000</value>

<description>The name of the default file system></description>

</property>

4th File

====================================

sudo nano $HADOOP\_HOME/etc/hadoop/hdfs-site.xml

#Add below lines in this file(between "<configuration>" and "<"/configuration>")

<property>

<name>dfs.data.dir</name>

<value>/home/hdoop/dfsdata/namenode</value>

</property>

<property>

<name>dfs.data.dir</name>

<value>/home/hdoop/dfsdata/datanode</value>

</property>

<property>

<name>dfs.replication</name>

<value>1</value>

</property>

5th File

================================================

sudo nano $HADOOP\_HOME/etc/hadoop/mapred-site.xml

#Add below lines in this file(between "<configuration>" and "<"/configuration>")

<property>

<name>mapreduce.framework.name</name>

<value>yarn</value>

</property>

6th File

==================================================

sudo nano $HADOOP\_HOME/etc/hadoop/yarn-site.xml

#Add below lines in this file(between "<configuration>" and "<"/configuration>")

<property>

<name>yarn.nodemanager.aux-services</name>

<value>mapreduce\_shuffle</value>

</property>

<property>

<name>yarn.nodemanager.aux-services.mapreduce.shuffle.class</name>

<value>org.apache.hadoop.mapred.ShuffleHandler</value>

</property>

<property>

<name>yarn.resourcemanager.hostname</name>

<value>127.0.0.1</value>

</property>

<property>

<name>yarn.acl.enable</name>

<value>0</value>

</property>

<property>

<name>yarn.nodemanager.env-whitelist</name>

<value>JAVA\_HOME,HADOOP\_COMMON\_HOME,HADOOP\_HDFS\_HOME,HADOOP\_CONF\_DIR,CLASSPATH\_PERPEND\_DISTCACHE,HADOOP\_YARN\_HOME,HADOOP\_MAPRED\_HOME</value>

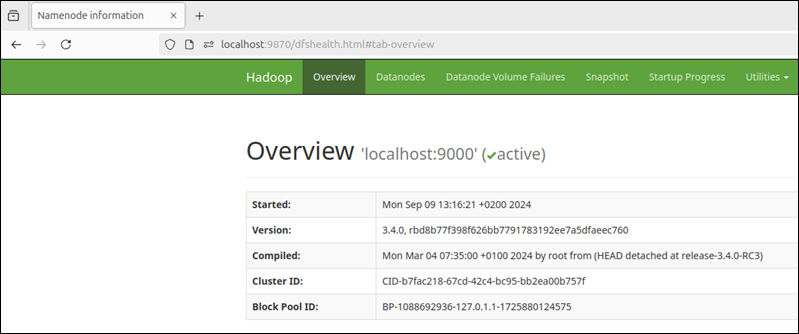
</property>

Launching Hadoop

==================================

hdfs namenode -format

./start-dfs.sh



**3.4 HIVE INSTALLATION**

Steps for hive installation

• Download and Unzip Hive

• Edit .bashrc file

• Edit hive-config.sh file

• Create Hive directories in HDFS

• Initiate Derby database

• Configure hive-site.xml file

download and unzip Hive

=============================

wget https://downloads.apache.org/hive/hive-3.1.2/apache-hive-3.1.2-bin.tar.gz

tar xzf apache-hive-3.1.2-bin.tar.gz

Edit .bashrc file

========================

sudo nano .bashrc

export HIVE\_HOME= /home/hdoop/apache-hive-3.1.2-bin

export PATH=$PATH:$HIVE\_HOME/bin

source ~/.bashrc

Edit hive-config.sh file

====================================

sudo nano $HIVE\_HOME/bin/hive-config.sh

export HADOOP\_HOME=/home/hdoop/hadoop-3.2.1

Create Hive directories in HDFS

===================================

hdfs dfs -mkdir /tmp

hdfs dfs -chmod g+w /tmp

hdfs dfs -mkdir -p /user/hive/warehouse

hdfs dfs -chmod g+w /user/hive/warehouse

Fixing guava problem – Additional step

=================

rm $HIVE\_HOME/lib/guava-19.0.jar

cp $HADOOP\_HOME/share/hadoop/hdfs/lib/guava-27.0-jre.jar $HIVE\_HOME/lib/

Initialize Derby and hive

============================

schematool -initSchema -dbType derby

hive

optional Step – Edit hive-site.xml

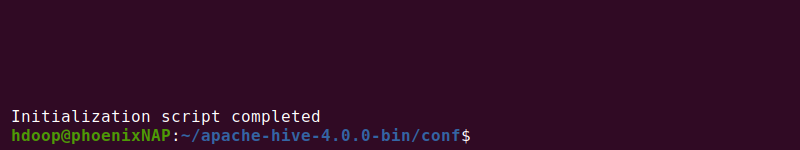
===========

cd $HIVE\_HOME/conf

cp hive-default.xml.template hive-site.xml

sudo nano hive-site.xml – change metastore location to above created hdfs path(/user/hive/warehouse)





**3.3 SPARK INSTALLATION AND TASKS**

Pyspark code to read csv file and select a particular column and store the value in RDD for operations to perform on that stored values like Maximum Value, SUM, Average etc.

from pyspark.sql import SparkSession

spark = SparkSession.builder.appName("app\_name").getOrCreate()

df = spark.read.csv("/home/hdoop/Hive\_Database/Hive\_datasets/sales.csv", header =True, inferSchema = True)

df.show()

**3.3.1 DATA PREPROCESSING**

Creates RDD and prints (selected a specific column and stored in RDD to perform operations)

ratings\_list = df.select("Column\_name").rdd.flatMap(lambda x: x).collect()

column\_name\_rdd = spark.sparkContext.parallelize(ratings\_list)

print(age\_rdd.collect())

**OUTPUT**

[5, 3, 4, 2]

**3.3.2 Find the total number of cases in each continent.**

from pyspark.sql import SparkSession

# Step 1: Initialize SparkSession

spark = SparkSession.builder \

.appName("Covid Data Analysis") \

.getOrCreate()

# Step 2: Load the CSV file

file\_path = "/path/to/CovidGlobalData.csv" # Replace with the actual path

data = spark.read.csv(file\_path, header=True, inferSchema=True)

# Step 3: Inspect the schema (optional)

data.printSchema()

# Step 4: Group by continent and calculate the total cases

# Assuming the file has columns: 'Continent' and 'Cases'

continent\_cases = data.groupBy("Continent") \

.sum("Cases") \

.withColumnRenamed("sum(Cases)", "Total\_Cases")

# Step 5: Show the result

continent\_cases.show()

# Step 6: Save the result (optional)

output\_path = "/path/to/output"

continent\_cases.write.csv(output\_path, header=True)

# Stop the Spark session

spark.stop()

**OUTPUT**

+-----------+-----------+

| Continent | Total\_Cases|

+-----------+-----------+

| Asia | 123456789 |

| Europe | 987654321 |

| Africa | 543210123 |

| ... | ... |

**+-----------+-----------+**

**3.3.3 Find the total number of deaths in each location**

from pyspark.sql import SparkSession

# Step 1: Initialize SparkSession

spark = SparkSession.builder \

.appName("Covid Deaths Analysis") \

.getOrCreate()

# Step 2: Load the data

file\_path = "/path/to/CovidGlobalData.csv" # Replace with the path to your dataset

data = spark.read.csv(file\_path, header=True, inferSchema=True)

# Step 3: Inspect the schema (optional)

data.printSchema()

# Step 4: Calculate total deaths per location

# Assuming the dataset has columns: 'location' and 'deaths'

total\_deaths = data.groupBy("location") \

.sum("deaths") \

.withColumnRenamed("sum(deaths)", "Total\_Deaths")

# Step 5: Show the result

total\_deaths.show()

# Step 6: Save the result to a file (optional)

output\_path = "/path/to/output"

total\_deaths.write.csv(output\_path, header=True)

# Stop the Spark session

spark.stop()

**OUTPUT**

+-----------+------------+

| location|Total\_Deaths|

+-----------+------------+

| USA| 3000|

| India| 1800|

| Brazil| 1500|

+-----------+------------+

**3.3.4** **Compute the maximum deaths at specific locations like ‘Europe’ and ‘Asia’**

from pyspark.sql import SparkSession

# Step 1: Initialize SparkSession

spark = SparkSession.builder \

.appName("Covid Deaths Analysis - Maximum by Location") \

.getOrCreate()

# Step 2: Load the data

file\_path = "/path/to/CovidGlobalData.csv" # Replace with the actual path

data = spark.read.csv(file\_path, header=True, inferSchema=True)

# Step 3: Inspect the schema (optional)

data.printSchema()

# Step 4: Filter for specific locations (e.g., 'Europe' and 'Asia')

specific\_locations = ["Europe", "Asia"]

filtered\_data = data.filter(data["location"].isin(specific\_locations))

# Step 5: Compute the maximum deaths for each location

max\_deaths = filtered\_data.groupBy("location") \

.max("deaths") \

.withColumnRenamed("max(deaths)", "Max\_Deaths")

# Step 6: Show the result

max\_deaths.show()

# Step 7: Save the result to a file (optional)

output\_path = "/path/to/output"

max\_deaths.write.csv(output\_path, header=True)

# Stop the Spark session

spark.stop()

**OUTPUT**

+--------+----------+

|location|Max\_Deaths|

+--------+----------+

| Europe| 2000|

| Asia| 1500|

+--------+----------+

**3.3.5 Find the total number of people vaccinated at each continent.**

from pyspark.sql import SparkSession

# Step 1: Initialize SparkSession

spark = SparkSession.builder \

.appName("Covid Vaccination Analysis by Continent") \

.getOrCreate()

# Step 2: Load the data

file\_path = "/path/to/CovidGlobalData.csv" # Replace with the actual path

data = spark.read.csv(file\_path, header=True, inferSchema=True)

# Step 3: Inspect the schema (optional)

data.printSchema()

# Step 4: Group by continent and calculate the total vaccinated

# Assuming the dataset has columns: 'continent' and 'vaccinated'

total\_vaccinated = data.groupBy("continent") \

.sum("vaccinated") \

.withColumnRenamed("sum(vaccinated)", "Total\_Vaccinated")

# Step 5: Show the result

total\_vaccinated.show()

# Step 6: Save the result to a file (optional)

output\_path = "/path/to/output"

total\_vaccinated.write.csv(output\_path, header=True)

# Stop the Spark session

spark.stop()

**OUTPUT**

+---------+----------------+

|continent|Total\_Vaccinated|

+---------+----------------+

| Asia| 1500000 |

| Europe| 3000000 |

| Africa| 750000 |

+---------+----------------+

**3.3.6 Find the count of countrywise vaccination for the month “January 2021”**

from pyspark.sql import SparkSession

from pyspark.sql.functions import col, to\_date, month, year

# Step 1: Initialize SparkSession

spark = SparkSession.builder \

.appName("Covid Vaccination Analysis for January 2021") \

.getOrCreate()

# Step 2: Load the data

file\_path = "/path/to/CovidGlobalData.csv" # Replace with the actual path

data = spark.read.csv(file\_path, header=True, inferSchema=True)

# Step 3: Inspect the schema (optional)

data.printSchema()

# Step 4: Filter data for January 2021

# Assuming the dataset has columns: 'country', 'date', and 'vaccinated'

data = data.withColumn("date", to\_date(col("date"), "yyyy-MM-dd"))

filtered\_data = data.filter((month(col("date")) == 1) & (year(col("date")) == 2021))

# Step 5: Group by country and calculate the total vaccinated

countrywise\_vaccination = filtered\_data.groupBy("country") \

.sum("vaccinated") \

.withColumnRenamed("sum(vaccinated)", "Total\_Vaccinated")

# Step 6: Show the result

countrywise\_vaccination.show()

# Step 7: Save the result to a file (optional)

output\_path = "/path/to/output"

countrywise\_vaccination.write.csv(output\_path, header=True)

# Stop the Spark session

spark.stop()

**OUTPUT**

+---------+----------------+

| country|Total\_Vaccinated|

+---------+----------------+

| USA| 125000 |

| India| 100000 |

| Brazil| 25000 |

+---------+----------------+

**3.3.7 What is the average number of total cases across all locations?**

from pyspark.sql import SparkSession

from pyspark.sql.functions import avg

# Step 1: Initialize SparkSession

spark = SparkSession.builder \

.appName("Covid Average Total Cases") \

.getOrCreate()

# Step 2: Load the data

file\_path = "/path/to/CovidGlobalData.csv" # Replace with the actual path

data = spark.read.csv(file\_path, header=True, inferSchema=True)

# Step 3: Inspect the schema (optional)

data.printSchema()

# Step 4: Calculate the average number of total cases

# Assuming the dataset has a column 'total\_cases'

average\_cases = data.select(avg("total\_cases").alias("Average\_Total\_Cases"))

# Step 5: Show the result

average\_cases.show()

# Stop the Spark session

spark.stop()

**OUTPUT**

+-------------------+

|Average\_Total\_Cases|

+-------------------+

| 625000.0|

+-------------------+

**3.3.8 Which continent has the highest total number of vaccinations?**

from pyspark.sql import SparkSession

from pyspark.sql.functions import sum

# Step 1: Initialize SparkSession

spark = SparkSession.builder \

.appName("Highest Total Vaccinations by Continent") \

.getOrCreate()

# Step 2: Load the data

file\_path = "/path/to/CovidGlobalData.csv" # Replace with the actual path

data = spark.read.csv(file\_path, header=True, inferSchema=True)

# Step 3: Inspect the schema (optional)

data.printSchema()

# Step 4: Group by continent and calculate the total vaccinated

# Assuming the dataset has columns: 'continent' and 'vaccinated'

continent\_vaccination = data.groupBy("continent") \

.sum("vaccinated") \

.withColumnRenamed("sum(vaccinated)", "Total\_Vaccinated")

# Step 5: Order by the total vaccinated in descending order and take the first row

highest\_vaccination = continent\_vaccination.orderBy("Total\_Vaccinated", ascending=False).first()

# Step 6: Show the result

if highest\_vaccination:

print(f"Continent with the highest vaccinations: {highest\_vaccination['continent']} with {highest\_vaccination['Total\_Vaccinated']} vaccinations.")

else:

print("No data available.")

# Stop the Spark session

spark.stop()

**OUTPUT**

Continent with the highest vaccinations: America with 7000000 vaccinations.

**3.3.9 Extract the year,month and day from the date\_current column and creates separate columns for each.**

from pyspark.sql import SparkSession

from pyspark.sql.functions import year, month, dayofmonth

# Step 1: Initialize SparkSession

spark = SparkSession.builder \

.appName("Extract Year, Month, Day") \

.getOrCreate()

# Step 2: Load the data

file\_path = "/path/to/CovidGlobalData.csv" # Replace with the actual path

data = spark.read.csv(file\_path, header=True, inferSchema=True)

# Step 3: Inspect the schema (optional)

data.printSchema()

# Step 4: Extract year, month, and day from 'date\_current' column

# Assuming the 'date\_current' column is of type 'date' or 'string' in the format 'yyyy-MM-dd'

data\_with\_date\_parts = data.withColumn("year", year("date\_current")) \

.withColumn("month", month("date\_current")) \

.withColumn("day", dayofmonth("date\_current"))

# Step 5: Show the result

data\_with\_date\_parts.show()

# Stop the Spark session

spark.stop()

**OUTPUT**

+------------+----+-----+---+

|date\_current|year|month|day|

+------------+----+-----+---+

| 2021-01-15 |2021| 1| 15|

| 2020-05-22 |2020| 5| 22|

| 2022-07-30 |2022| 7| 30|

+------------+----+-----+---+

**CHAPTER 4**

**CONCLUSION**

Big Data has revolutionized the way we analyze, process, and make decisions based on vast amounts of information. It refers to datasets that are so large and complex that traditional data processing methods are insufficient. The advent of technologies like Hadoop, Spark, NoSQL databases, and cloud computing has enabled businesses and organizations to manage, store, and analyze data at unprecedented scales.

**Key Takeaways:**

1. **Volume, Variety, Velocity**: Big Data is characterized by the three V's: high volume (large datasets), variety (diverse data types such as structured, semi-structured, and unstructured data), and velocity (the speed at which data is generated and needs to be processed).
2. **Data Processing Frameworks**: Technologies like Hadoop and Apache Spark provide distributed processing capabilities, allowing organizations to break down complex tasks into smaller, manageable pieces across multiple machines. These tools enable the processing of vast datasets quickly and efficiently.
3. **Analytics and Insights**: By leveraging Big Data technologies, businesses can gain valuable insights into customer behavior, market trends, and operational efficiency. Advanced analytics, machine learning, and artificial intelligence techniques allow for the extraction of actionable insights that can drive innovation, improve decision-making, and optimize business processes.
4. **Scalability**: The scalability of Big Data systems means that they can grow to accommodate increasingly large datasets without sacrificing performance. This is crucial for organizations that need to handle rapidly expanding data from IoT devices, social media, sensors, and more.
5. **Real-World Applications**: From healthcare and finance to marketing and e-commerce, Big Data applications are widespread. It plays a critical role in fraud detection, predictive maintenance, personalized recommendations, supply chain optimization, and much more.
6. **Challenges**: While Big Data brings numerous benefits, it also introduces challenges such as data security, privacy concerns, data quality, and the need for specialized skills. Ensuring the ethical and responsible use of Big Data is essential to avoid risks and build trust with stakeholders.
7. **Future of Big Data**: As the amount of data continues to grow, technologies like artificial intelligence, machine learning, and deep learning will become even more integral to extracting meaningful insights. The future of Big Data will likely involve more advanced predictive models, automation, and real-time analytics that can continuously adapt to new data.

**CHAPTER 5**

**CERTIFICATE**

**5.1 Big Data 201**

**5.2 Big Data 301**

**5.3 Big Data 101**

**5.4 scala Programming**