UNIT 7 OTHER BIG DATA ARCHITECTURES AND TOOLS

Structure

7.0 INTRODUCTION

In the Unit 5 and Unit 6 of this Block, you have gone through the concepts of Hadoop and MapReduce programming. In addition, you have gone through various phases in the MapReduce program. This unit introduces you to other popular big data architecture tools and architectures. These architectures are beneficial to both the ETL developers and analytics professionals.

This Unit introduces you to the basic software stack of SPARK architecture, one of the most popular architecture of handling large data. In addition, this Unit introduces two important tools – HIVE, which is a data warehouse system and HBase, which is an important database system. An open-source NoSQL database called HBase uses HDFS and Apache Hadoop to function. Unlimited data can be stored on this extendable storage. Built on HDFS, HIVE is a SQL engine that uses MapReduce.

7.1 OBJECTIVES

After completing this unit, you will be able to:

- list the fundamentals of big data processing using Spark ecosystem;
- explain the main components of Spark framework;
- define the fundamental components of Hive;
- installation guidelines for these tools: Spark, Hive and HBase
- illustrate the query processing in Hive and HBase;

7.2 APACHE SPARK FRAMEWORK

With SPARK, the scalability and fault tolerance of Hadoop MapReduce were preserved while being designed for quick iterative processing, such as machine learning, and interactive data analysis. As we have already discussed in Unit 6, the MapReduce programming model, which is the foundation of the Hadoop framework, allows for scalable, adaptable, fault-resilient, and cost friendly solutions. In this case, it becomes imperative to reduce the turnaround time between queries and execution. The Apache Software Foundation released SPARK to quicken the Hadoop computing processes. SPARK has its own cluster management, hence it is not dependent on Hadoop and is not a revised form of Hadoop. Hadoop is merely one method of implementing SPARK. SPARK's in-memory cluster computing, which accelerates application processing, is its key feature. Numerous tasks, including batch processing, iterative algorithms, interactive queries, and streaming, can be handled by SRARK. Along with accommodating each job in its own system, it lessens the administrative strain of managing various tools.

SPARK is a data processing framework, which can swiftly conduct operations on huge datasets and distribute operations across several computers, either alone as well as in conjunction with other distributed computing tools. Above characteristics are essential to in big data and machine learning computing, which both call for immense computing resources to be mobilised in order to process enormous data stored in data warehouses. Spark provides a user-friendly Application Programming Interface (API), which abstracts away a lot of the tedious programming activities involved in big data processing and distributed computing, relieving developers of several coding related functionality associated with these activities. Spark framework has been constructed over the Hadoop MapReduce platform and expands the MapReduce framework to be utilised for other calculations, including computation of dynamic queries and stream processing, more effectively.

In 2009, UC Berkeley based AMPLab launched Apache Spark as a research initiative with an emphasis on data-intensive application areas and further got open sourced in the year 2010 with BSD License. This partnership involved students, researchers, and professors. The main features of Apache Spark framework includes:

i) Fast processing: In a Hadoop cluster, Spark makes it possible for applications to execute almost 100 times quicker in memory and 10 times on storage by reducing the amount of read/write operations.
ii) Multi-language support: Python, Scala, or Java-based built-in APIs are available with Spark, and
iii) Advance Analytics support: Spark offers more than just "Map phase" and "Reduce phase." Additionally, it also enables support for graph methods, data streaming data, machine learning (ML), and SQL queries.

Figure 1 depicts the Spark data lake and shows how the Apache Spark can work in conjunction with Hadoop components and information flow happens with

Apache Spark. Hadoop Distributed File Storage (HDFS) allows us to form cluster of computing machines and utilize the combined capacity to store the data, thus allowing to store huge data volumes. Further, MapReduce allows to use combined power of the clusters and process it to store enormous data stored in HDFS. The advent of HDFS and MapReduce allows horizontal scalability and low capital cost as compared to data warehouses. Gradually, cloud infrastructure became more economical and got wider adoption. Large amounts of organised, semi-structured, and unstructured data can be stored, processed, and secured using a data lake, a centralised repository. Without regard to size restrictions, it can process any type of data and store it in its native format. A data lake has four key capabilities: i) Ingest: allows data collection and ingestion, ii) Store: responsible for data storage and management, iii) Process: leads to transformation and data processing, and iv) Consume: ensures data access and retrieval. In store capability of data lake, it could be an HDFS or cloud store such as in Amazon S3, Azure Blob, Google cloud storage etc. The cloud storage allows scalability and high availability access at an extremely low cost in almost no time to procure. The notion of the data lake recommends to bring data into data lake in raw format, i.e. ingest data into data lake and preserve an unmodified immutable copy of data. The ingest block of data lake is about identifying, implementing and managing the right tools to bring data from the source system to the data lake. There is no single ingestion tools thus there could be multiple tools such as HVR, Informatica, Talend etc. The next layer is the process layer where all computation takes place such as initial data quality check, transforming and preparing data, correlating, aggregating, and analysing and applying machine learning models. Processing layer is further broken into two parts which helps to manage better: i) data processing and ii) orchestration. The processing is the core development framework allows to design and develop distributed computing frameworks. Apache Spark is part of data processing. The orchestration framework is responsible for the formation of the clusters, managing resources, scaling up or down etc. There are three main competing orchestration tools such as Hadoop Yarn, Kubernetes and Apache Mesos. The last and most critical capability of data lake is to consume the data from data lake for real life usage. The data lake is a repository of raw and processed data. The consumption requirements could be from data analysts/scientists, or from some applications or dashboards to take insights from data, or from JDBC/ODBC connectors, others might be from Rest interface etc.

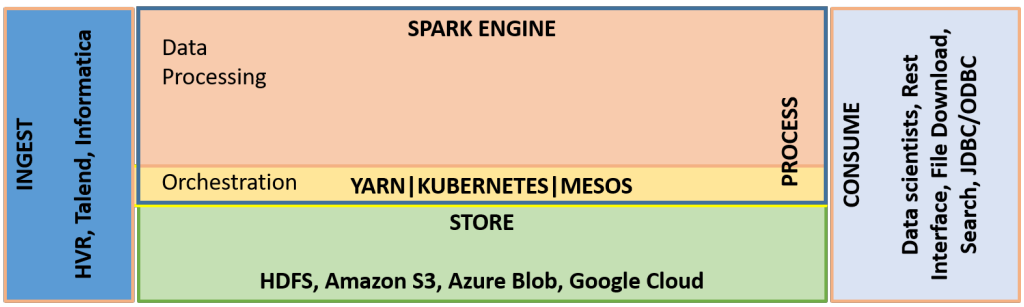


Figure 1: SPARK data lake

Apache Spark deployment can happen in three modes:

i) Standalone: When Spark is deployed independently, space is set aside for HDFS (the Hadoop Distributed File System) on top of Spark.

Spark and MapReduce will coexist in this scenario to handle all Spark jobs on the cluster.

ii) Hadoop Yarn: Hadoop yarn deployment means that spark runs on Yarn without the need for any prior installation or root access. Integrating Spark within the Hadoop stack is beneficial. It enables running of other components on top of stack.

iii) Spark in MapReduce: Along with standalone deployment, Spark is used in MapReduce to start spark jobs. Without requiring administrative access, a user can launch Spark and use its shell using Spark in MapReduce.

The main components of Apache Spark (as shown in Figure 2) are as follows:

i) Spark Core: All other functionality for the Spark platform is built upon the universal execution engine known as Spark Core. It offers in-memory computation (i.e. running complex calculations entirely in computer memory as in RAM) and refers to datasets in external storage systems. It oversees memory management, fault recovery, job scheduling, job distribution, and job monitoring, as well as communication with storage systems. APIs that are created for Java, Scala, Python, and R make Spark Core accessible. These APIs leverage the complex operations of distributed processing with straightforward, high-level operators.

ii) Spark SQL: On top of Spark Core, the Spark SQL component adds a new data abstraction called SchemaRDD that supports both structured and semi-structured data. Row objects with a schema describing the data types of each column in the row are combined to form SchemaRDDs. A table in a conventional relational database is analogous to a SchemaRDD. Resilient Distributed Datasets (RDD) and schema information are combined to generate SchemaRDD. The logical arrangement of structured data is described by a schema. For quick access to data during computation and fault tolerance, the RDDs store data in memory. A distributed query engine called Spark SQL offers interactive query processing up to 100 times quicker as compared to MapReduce. It scales to thousands of nodes and features a less-cost optimizing engine, tabular arrangement, and interactive query generation. For data queries, business analysts can utilise either traditional SQL or the Hive Query Language. APIs are accessible in Scala, Java, Python, and R for developers.

iii) Spark Streaming: It is used for analysis if data streams. This component makes use of the fast scheduling technique available in the Spark Core. It processes RDD (Resilient Distributed Datasets) modifications on the mini-batches of data that it ingests. Spark Streaming works towards real-time streaming analytics. Blocks of data that arrive in certain time intervals are taken by Spark Streaming and packaged as RDDs. A Spark Streaming job can receive data from numerous external services. These comprise TCP/IP socket connections and filesystems in addition to other distributed systems like Kafka, Flume, Twitter, and Amazon Kinesis. Receivers are capable of establishing a connection to the source, reading the data, and sending it to Spark Streaming. After splitting the incoming data into mini-batch RDDs—one mini-batch RDD for each time period—Spark Streaming allows the Spark application to process the data. The outcomes of computations can be stored in relational databases, filesystems, or other distributed systems.

iv) MLib (Machine Learning Library): It is a distributed framework for machine learning processing. According to the benchmarks, it has been tested against Alternating Least Squares (ALS) implementations by the MLlib developers. Spark MLlib is 9 times faster than Apache Mahout in a Hadoop disk-based variant. In Hadoop clusters, the Mahout library serves as the primary machine learning platform. Mahout uses MapReduce to carry out classification, clustering, and recommendation. Hadoop MapReduce divides tasks into parallel ones, some of which may be too big for machine learning algorithms. These Hadoop applications experience input-output performance difficulties as a result of this approach. Data scientists can use R or Python to train machine learning models on any Hadoop data source, save the models and then import the models into a Java- or Scala-based pipeline for production purposes. Spark can be used to perform machine learning tasks faster. It may be noted that Spark is very useful for quick, interactive computing that operates in memory. Thus, you can perform regression, classification, collaborative filtering, clustering, and pattern mining using Spark.

v) GraphX: It is a distributed graph-processing framework. It offers a graph computing API that uses the Pregel abstraction API to model user-defined graphs. Since, in graph data structure vertices are heavily dependent on the neighbour properties they exhibit recursion. As a result, many significant graph algorithms repeatedly compute each vertex's characteristics until a fixed-point condition is met. These iterative algorithms have been expressed using a variety of graph-parallel abstractions. Additionally, it offers a runtime that is optimised for this abstraction. In order to let the users build interactively and alter the structure of the graphs, GraphX offers ETL, exploratory analysis, and iterative graph computation. It comes with a variety of distributed Graph algorithms and a very versatile API.

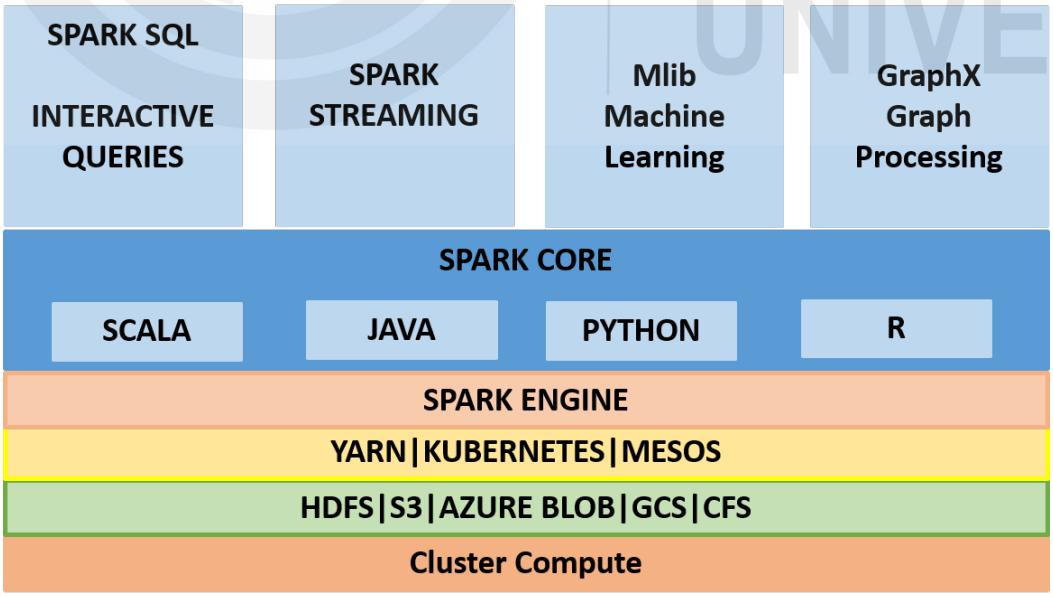


Figure 2: Apache Spark Ecosystem

The main data structure is Resilient Distributed Datasets (RDD) in Spark framework. It is a collection of immutable objects. Each data-set in an RDD is split into logical partitions that can be computed on several cluster nodes. An

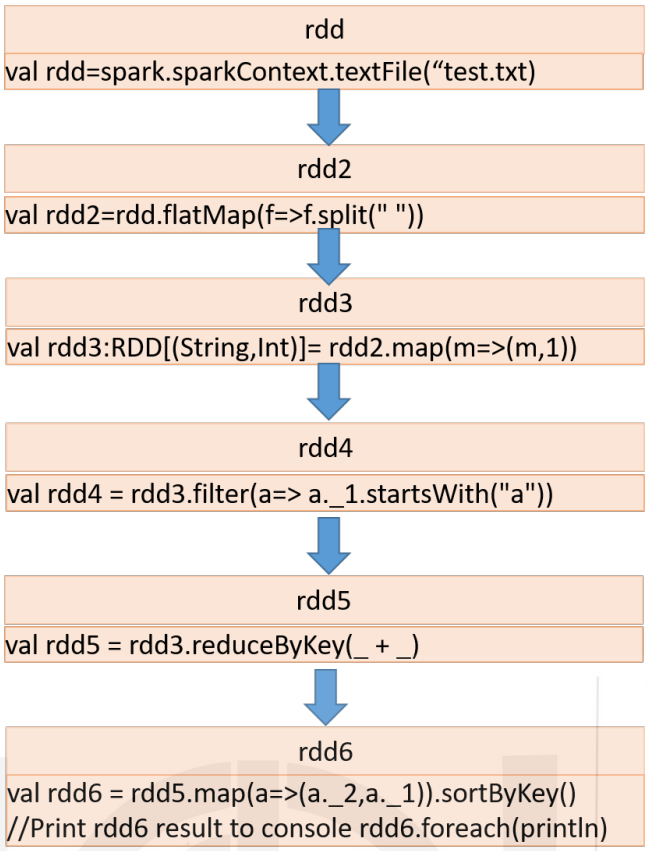
RDD is formally a partitioned, collection of records in read only format. RDDs can be created due to fixed operations on either the data on stable storage (disk) or other RDDs. RDD is a set of fault-tolerant components that can be used concurrently. Normally text files, SQL databases, NoSQL stores, Amazon S3 buckets, and many other sources can all be used to construct RDDs. This RDD idea serves as the foundation for a significant portion of the Spark Core API, enabling not only the standard map and reduce functionality but also built-in support for merging data sets, and operations such as filter, sample, and aggregate data. By combining a driver core process, which divides a Spark application into various jobs and distributes them across numerous executor processes that carry out the job, Spark works in a distributed manner. These executor processes can be increased up or down depending on the demands of the application. Spark uses the RDD paradigm to expedite and optimise MapReduce operations. RDD Transformations are lazy operations, they are not started until you invoke an action on the Spark RDD. RDDs are immutable, therefore any transformations on them produce a new RDD while maintaining the original's integrity. There are two types of transformations: i) Narrow: These are result of map() and filter() functions and work on data in single partition i.e. there will not be any data movement amongst partitions in narrow transformations, and ii) Wider: These are results of groupbykey() and reducebykey() i.e. it will work on data from multiple partitions and allow data movement amongst them. Let us consider the RDD transformations in word count example:

```
val spark:SparkSession = SparkSession.builder()
    .master("local[3]")
    .appName("SparkByExamples.com")
    .getOrCreate()

val sc = spark.sparkContext

val rdd:RDD[String] = sc.textFile("src/main/scala/test.txt")
```

The set of RDD transformations in word count example is as follows:



Here, we first create an RDD by reading any input text file namely "test.txt". After applying the function, the flatMap() transformation flattens the RDD and creates a new RDD. It first divides each record in an RDD by space before flattening it. Each entry in the resulting RDD only contains one word. Any complex actions, such as the addition of a column or the updating of a column, are applied using the map() transformation, and the output of these transformations always has the same amount of records as the input. In this example, we are creating a new column and assigning a value of 1 to each word. The RDD produces a PairRDDFunction that has key-value pairs with the keys being words of type String and the values being 1 of type Int. The records in an RDD can be filtered using the filter() transformation. In the example, we are filtering out all terms that begin with "a". reduceByKey() combines the specified function with the values for each key. Here, it shortens the word string by using sum function "+" on the value. The output of our RDD includes the number of unique words. RDD elements are sorted on keys using the sortByKey() transformation. In this example, we first use the map transformation to change RDD[(String,Int)] to RDD[(Int,String]] before using sortByKey, which ideally sorts on an integer value. Finally, the foreach with println statement outputs to the console all words in RDD and their count (as a key-value).

Installation Steps for Apache Spark framework (on Ubuntu):
Step 1: Check whether JAVA is installed or not, if not you need to install it.

```
$ java -version
```

Step 2: Check whether SCALA is installed or not since Spark is written in Scala, although elementary knowledge in Scala is enough to run Spark. Other supported languages in Spark are Java, R, Python.

```
$scala -version
```

In case it is not installed, you need to download it and
Extract the tar file

```
$ tar xvf scala-2.11.6.tgz
```

We need to move SCALA software files to /usr/ directory as follows:

```
$ su – Password: # cd /home/Hadoop/Downloads/ # mv scala-2.11.6 /usr/local/scala # exit
```

We need to set the environment path variable:

```
$ export PATH = $PATH:/usr/local/scala/bin
```

Last, check if it is installed successfully:

```
$scala -version
```

Step 3: Download Apache Spark
Step 4: Install Spark
We need to extract the requisite tar file

```
$ tar xvf spark-1.3.1-bin-hadoop2.6.tgz
```

We need to move SPARK software files to /usr/ directory as follows:

```
$ su – Password: # cd /home/Hadoop/Downloads/ # mv spark-1.3.1-bin-hadoop2.6 /usr/local/spark # exit
```

We need to set the environment path variable:

```
export PATH=$PATH:/usr/local/spark/bin
```

```
$ source ~/.bashrc
```

Last, check if SPARK is installed successfully:

```
$spark-shell
```

It will be installed successfully.

Check Your Progress 1:

1. What are the main features of Apache Spark Framework?
2. What are the components of Apache Spark Framework?
3. List and perform the steps of installation of Apache Spark.

Next, we discuss a data warehouse system of this architecture named HIVE system.

7.3 HIVE

A Hadoop utility for processing structured data is called Hive. It sits on top of Hadoop to summarize big data and simplifies querying and analysis. Initially created by Facebook, Hive was later taken up and further developed as an open source project under the name Apache Hive by the Apache Software Foundation. It is utilised by various businesses. Hive is not a relational database like SQL. HIVE does not support for “Row-level” modifications of data, as the case in SQL based database management systems. It also allows real time queries on data. It puts processed data into HDFS and stores the schema in a database. It offers a querying language called HiveQL or HQL that is similar to SQL. It is dependable, quick, expandable, and scalable.

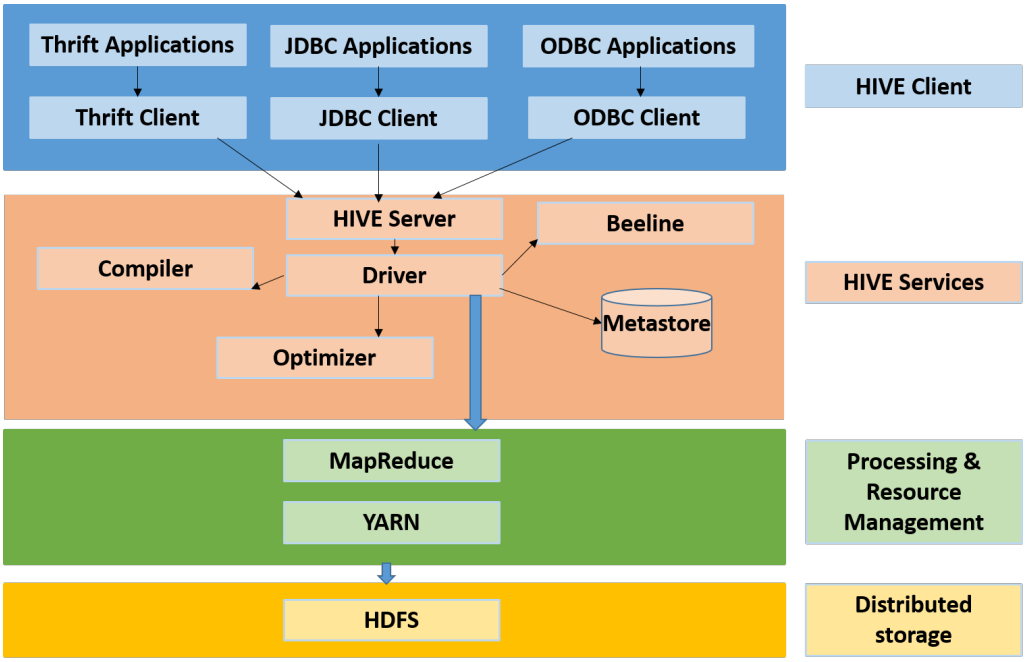


Figure 3: HIVE Architecture

The main components in Hive architecture (as shown in Figure 3) can be categorized into:

i) Hive Client: Hive client allows to connect to Hive data with the help of database drivers like ODBC, JDBC, etc. Hive enables applications built in any of programming languages including C++, Ruby, JAVA, Python etc. to be used as Hive client. Thus, creating an application for the hive client in any language of one's preference is quite simple. These clients can be of three types:

a) Thrift Clients: Since Apache Thrift is the foundation of the Hive server, it may fulfil requests from thrift clients. Thrift is a communication protocol used by applications created in many programming languages. As a result, client programmes send requests or hive queries to the Thrift server, which is located in the hive services layer. b) JDBC Clients: Java applications can be connected with Hive by utilising the JDBC driver. Thrift is used by the JDBC driver to interact with the Hive Server, and c) ODBC Clients: ODBC based applications can be connected with Hive using the Hive ODBC driver. It communicates with the Hive Server using Thrift, just like the JDBC driver does.

ii) Hive services: Hive provides a range of services for various purposes. The following are some of the most useful services:

a. Beeline: It is a command shell that HiveServer2 supports, allowing users to send commands and queries to the system. It is JDBC Client based on SQLLINE Command Line Interface (CLI). SQLLINE CLI is a Java Console-based utility for running SQL queries and connecting to relational databases.

b. Hive Server 2: After the popularity of HiveServer1, next HiveServer2 was launched which helps the clients to run the queries. It enables numerous clients to ask Hive questions and get query responses.

c. Hive Driver: The user submits a Hive query using HiveQL statements via the command shell, which are then received by the Hive driver. The query is then sent to the compiler where session handles for the query that are created.

d. Hive Compiler: The query is parsed by Hive compiler. The metadata of the database, which is stored in the metastore can be used to perform semantic analysis and data type validation on the various parsed queries, and then it provides an execution plan. The DAG (Directed Acyclic Graph) is the execution plan the compiler generates, with each stage referred as a map or a reduce job, HDFS action, and metadata operation.
e. Optimizer: To increase productivity and scalability, the optimizer separates the work and performs transformation actions on the execution plan so as to optimise the query execution time.
f. Execution engine: Following the optimization and compilation processes, the execution engine uses Hadoop to carry out the execution plan generated by the compiler as per the dependency order amongst the execution plan.
g. Metastore: The metadata information on the columns and column types in tables and partitions is kept in a central location called the metastore. Additionally, it stores data storage information for HDFS files as well as serializer and deserializer information needed for read or write operations. Typically, this metastore is a relational database. A Thrift interface is made available by Metastore for querying and modifying Hive metadata.
h. HCatalog: It refers to the storage and table management layer of Hadoop. It is constructed above metastore and makes Hive metastore's tabular data accessible to other data processing tools. And
i. WebHCat: HCatalog's REST API is referred to as WebHCat. A Hadoop table storage management tool called HCatalog allows other Hadoop applications to access the tabular data of the Hive metastore. A web service architecture known as REST API is used to create online services that communicate via the HTTP protocol. WebHCat is an HTTP interface for working with Hive metadata.

iii) Processing and Resource Management: Internally, the de facto engine for Hive's query execution is the MapReduce framework, which is a software framework. MapReduce is used to create map and reduce functions that process enormous amounts of data concurrently on vast clusters of commodity hardware. Data is divided into pieces and processed by map-reduce tasks as part of a map and reduce jobs.

iv) Distributed Storage: Since Hadoop is the foundation of Hive, the distributed storage is handled by the Hadoop Distributed File System.

7.3.1 Working of HIVE Queries

There are various steps involved in the execution of Hive queries as follows:
Step 1: executeQuery Command: Hive UI either command line or web UI sends the query which is to be executed to the JDBC/ODBC driver.
Step 2: getPlan Command: After accepting query expression and creating a handle for the session to execute, the driver instructs the compiler to produce an execution plan for the query.
Step 3: getMetadata Command: Compiler contacts the metastore with a metadata request. The hive metadata contains the table's information such as schema and location and also the information of partitions.

Step 4: sendMetadata Command: The metastore transmits the metadata to the compiler. These metadata are used by the compiler to type-check and analyse the semantics of the query expressions. The execution plan (in the form of a Directed Acyclic graph) is then produced by the compiler. This plan can use MapReduce programming. Therefore, the map and reduce jobs would include the map and the reduce operator trees.
Step 5: sendPlan Command: Next compiler communicates with the driver by sending the created execution plan.
Step 6: executePlan Command: The driver transmits the plan to the execution engine for execution after obtaining it from the compiler.
Step 7: submit job to MapReduce: The necessary map and reduce job worker nodes get these Directed Acyclic Graphs (DAG) stages after being sent by the execution engine. Each task, whether mapper or reducer, reads the rows from HDFS files using the deserializer. These are then handed over through the associated operator tree. As soon as the output is ready, the serializer writes it to the HDFS temporary file. The last temporary file is then transferred to the location of the table for Data Manipulation Language (DML) operations.
Step 8-10: sendResult Command: The execution engine now receives the temporary files contents directly from HDFS as a retrieve request from the driver. Results are then sent to the Hive UI by the driver.

7.3.2 Installation of HIVE

The step-by-step installation of Hive (on Ubuntu or any other Linux platform) is as follows:

Step 1: Check whether JAVA is installed or not, if not you need to install it.

```
$ java -version
```

Step 2: Check whether HADOOP is installed or not, if not you need to install it.

```
$ hadoop version
```

Step 3: You need to download Hive and verify the download:

```
$ cd Downloads $ ls
```

Step 4: Extract the Hive archive

```
$ tar zxvf apache-hive-0.14.0-bin.tar.gz $ ls
```

Step 5: Copy the files to /usr/ directory

```
$ su - passwd: # cd /home/user/Download # mv apache-hive-0.14.0-bin /usr/local/hive # exit
```

Step 6: Set up environment for Hive by adding the following to ~/.bashrc file:

```
export HIVE_HOME=/usr/local/hive
export PATH=$PATH:$HIVE_HOME/bin
export CLASSPATH=$CLASSPATH:/usr/local/Hadoop/lib/*:.
export CLASSPATH=$CLASSPATH:/usr/local/hive/lib/*:.
```

To execute ~/.bashrc file

```
$ source ~/.bashrc
```

Step 7: Configure Hive and edit hive-env.sh file

```
$ cd $HIVE_HOME/conf $ cp hive-env.sh.template hive-env.sh
```

```
export HADOOP_HOME=/usr/local/hadoop
```

Hive installation is complete. To configure metastore, we need an external database server such as Apache Derby.

Step 8: Download Derby:

```
$ cd ~ $ wget http://archive.apache.org/dist/db/derby/db-derby-10.4.2.0/db-derby-10.4.2.0-bin.tar.gz
```

Verify download by:

```
$ ls
```

After successful download, extract the archive:

```
$ tar zxvf db-derby-10.4.2.0-bin.tar.gz $ ls
```

Then, copy files to /usr/ directory

```
$ su - passwd: # cd /home/user # mv db-derby-10.4.2.0-bin /usr/local/derby # exit
```

Add these lines to ~/.bashrc file:

```
export DERBY_HOME=/usr/local/derby
export PATH=$PATH:$DERBY_HOME/bin
Apache Hive
18
export CLASSPATH=$CLASSPATH:$DERBY_HOME/lib/derby.jar:$DERBY_HOME/lib/derbytools.jar
```

To execute ~/.bashrc file

```
$ source ~/.bashrc
```

Create directory to store data in Metastore:

```
$ mkdir $DERBY_HOME/data
```

Step 9: Configure Metastore by editing hive-site.xml:

```
$ cd $HIVE_HOME/conf $ cp hive-default.xml.template hive-site.xml
```

Add these lines:

```
<property>
<name>javax.jdo.option.ConnectionURL</name>
<value>jdbc:derby://localhost:1527/metastore_db;create=true </value>
<description>JDBC connect string for a JDBC metastore </description>
</property>
```

Add these lines in file jpox.properties:

```
javax.jdo.PersistenceManagerFactoryClass =

org.jpox.PersistenceManagerFactoryImpl
org.jpox.autoCreateSchema = false
org.jpox.validateTables = false
org.jpox.validateColumns = false
org.jpox.validateConstraints = false
org.jpox.storeManagerType = rdbms
org.jpox.autoCreateSchema = true
org.jpox.autoStartMechanismMode = checked
org.jpox.transactionIsolation = read_committed
javax.jdo.option.DetachAllOnCommit = true
javax.jdo.option.NontransactionalRead = true
javax.jdo.option.ConnectionDriverName = org.apache.derby.jdbc.ClientDriver
javax.jdo.option.ConnectionURL = jdbc:derby://hadoop1:1527/metastore_db;create = true
 javax.jdo.option.ConnectionUserName = APP
 javax.jdo.option.ConnectionPassword = mine
```

Step 10: Hive installation verify:
The /tmp folder and a unique Hive folder must be created in HDFS before executing Hive. The /user/hive/warehouse folder is used here. For these recently formed directories, you must specify write permission as indicated below:

```
chmod g+w

$ $HADOOP_HOME/bin/hadoop fs -mkdir /tmp
$ $HADOOP_HOME/bin/hadoop fs -mkdir /user/hive/warehouse
$ $HADOOP_HOME/bin/hadoop fs -chmod g+w /tmp
$ $HADOOP_HOME/bin/hadoop fs -chmod g+w /user/hive/warehouse

$ cd $HIVE_HOME
$ bin/hive
```

Hive is successfully installed.

7.3.3 Writing Queries in HIVE

In order to write HIVE queries, you must install the HIVE software on your system. HIVE queries can be written using the query language that can be supported by Hive. THE different data types supported by Hive are given below.

Hive Data Types: Hive has 4 main data types:

1. Column Types: Integer, String, Union, Timestamp
2. Literals: Floating, Decimal
3. Null Values: All missing values as NULL
4. Complex Types: These complex types includes Arrays, Maps and struct. Syntax of these is given below:
   Arrays

   ```
   Syntax: ARRAY<data_type>
   ```

   Maps

   ```
   Syntax: MAP<primitive_type, data_type>
   ```

Struct

```
Syntax: STRUCT<col_name : data_type [COMMENT col_comment], ..
```

Hive Query Commands: Following are some of the basics commands to create and drop database, creating, dropping and altering tables; creating partitions etc. This section also lists some of the commands to query these database including join command

1. Create Database

```
hive> CREATE DATABASE [IF NOT EXISTS] userdb;
```

```
hive> SHOW DATABASES;
default
userdb
```

2. Drop database:

```
DROP DATABASE StatementDROP (DATABASE|SCHEMA) [IF EXISTS] da
[RESTRICT|CASCADE];
```

```
hive> DROP DATABASE IF EXISTS userdb;
```

```
hive> DROP DATABASE IF EXISTS userdb CASCADE;
```

Cascade helps to drop tables before dropping the database.

3. Creating Table:

```
CREATE [TEMPORARY] [EXTERNAL] TABLE [IF NOT EXISTS] [db_name

[(col_name data_type [COMMENT col_comment], ...)]
[COMMENT table_comment]
[ROW FORMAT row_format]
[STORED AS file_format]
```

4. Altering Table:

```
ALTER TABLE name RENAME TO new_name
ALTER TABLE name ADD COLUMNS (col_spec[, col_spec ...])
ALTER TABLE name DROP [COLUMN] column_name
ALTER TABLE name CHANGE column_name new_name new_type
ALTER TABLE name REPLACE COLUMNS (col_spec[, col_spec ...])
```

5. Dropping Table:

```
DROP TABLE [IF EXISTS] table_name;
```

6. Add Partition:

```
hive> ALTER TABLE employee
> ADD PARTITION (year='2012')
> location '/2012/part2012';
```

7. Operators Used:
   Relational Operators: =, !=, <, <=, >, >=, Is NULL, Is Not NULL, LIKE (to compare strings)
   Arithmetic Operators: +, -, *, /, %, & (Bitwise AND), | (Bitwise OR), ^ (Bitwise XOR), ~(Bitwise NOT)
   Logical Operators: &&, || , !
   Complex Operators: A[n] (nth element of Array A), M[key] (returns value of the key in map M), S.x (return x field of struct S)
8. Functions: round(), floor(), ceil(), rand(), concat(string M, string N), upper(), lower(), to_date(), cast()
   Aggregate functions: count(), sum(), avg(), min(), max()
9. Views:

```
CREATE VIEW [IF NOT EXISTS] view_name [(column_name [COMMENT column_comment], ...) ]
[COMMENT table_comment]
AS SELECT ...
```

10. Select Where Clause:
    It is used to retrieve a particular set of data from table. Where clause is the condition statement and gives finite result if condition is met.

```
SELECT [ALL | DISTINCT] select_expr, select_expr, ...
FROM table_reference
[WHERE where_condition]
[GROUP BY col_list]
[HAVING having_condition]
[CLUSTER BY col_list | [DISTRIBUTE BY col_list] [SORT BY col_list]]
[LIMIT number];
```

11. Select Order By Clause: To get information from a single column and sort the result set in either ascending or descending order, use the ORDER BY clause.

```
SELECT [ALL | DISTINCT] select_expr, select_expr, ...
FROM table_reference
[WHERE where_condition]
[GROUP BY col_list]
[HAVING having_condition]
[ORDER BY col_list]]
[LIMIT number];
```

12. Select Group By Clause: A result set can be grouped using a specific collection column by utilising the GROUP BY clause. It is used to search a collection of records.

```
SELECT [ALL | DISTINCT] select_expr, select_expr, ...
FROM table_reference
[WHERE where_condition]
[GROUP BY col_list]
[HAVING having_condition]
[ORDER BY col_list]]
[LIMIT number];
```

13. Join: The JOIN clause is used to combine specified fields from two tables using values that are shared by both. It is employed to merge data from two or more database tables.

```
join_table:

    table_reference JOIN table_factor [join_condition]
    | table_reference {LEFT|RIGHT|FULL} [OUTER] JOIN table_refe
    join_condition
    | table_reference LEFT SEMI JOIN table_reference join_condi
    | table_reference CROSS JOIN table_reference [join_conditio
```

Thus, most of the commands, as can be seen are very close to SQL like syntax. In case, you know SQL, you would be able to write the Hive queries too.

Check Your Progress 2:

1. What are the main components of HIVE architecture?
2. What are the main data types in HiveQL?
3. What are the different join types in HiveQL?

7.4 HBASE

RDBMS has been the answer to issues with data maintenance and storage since 1970s. After big data became prevalent, businesses began to realise the advantages of processing large data and began choosing solutions like Hadoop. Hadoop processes huge data using MapReduce and stores it in distributed file systems. Hadoop excels at processing and storing vast amounts of data in arbitrary, semi-structured, and even unstructured formats. Hadoop is only capable of batch processing, and only sequential access is possible to data. That implies that even for the most straightforward tasks, one must explore the entire dataset. When a large dataset is handled, it produces another equally large data collection, which should also be processed in a timely manner. At this stage, a fresh approach is needed to handle any data in a one go i.e. access at random.

On top of the Hadoop, the distributed column based data store HBase was created. It is a horizontally scalable open-source project. Similar to Google's Bigtable, HBase is a data model created to offer speedy random access to enormous amounts of structured data. It makes use of the Hadoop File System's fault tolerance (HDFS). It offers real-time read or write operations to access data randomly from the Hadoop File System. Data can be directly stored in HDFS or indirectly using HBase. Using HBase, data consumers randomly read from and access the data stored in HDFS. HBase is built on top of Hadoop file system, which offers read and write access. In nutshell, as compared to HDFS, HBase

enables faster lookup, random access with low latency due to internal storage in hash tables. The tables in the column-oriented database HBase are sorted by row (as depicted in Figure 4). The column families constituted of key-value pairs are defined by the table structure. A table is a grouping of rows. A row is made up of different column families. A collection of columns is called a column family. Each column has a set of key-value pairs.

Rowid	Column Family			Column Family			Column Family		
	col1	col2	col3	col1	col2	col3	col1	col2	col3
1									
2									
3									

Figure 4: Database schema of an HBase table

As compared to RDBMS, HBase is schema-less, meant for wide tables, has denomalized data, while cannot handle transactions. Other features of HBase includes:

- HBase scales linearly.
- It supports automatic failure.
- It offers reliable reading and writing.
- It is both a source and a destination of Hadoop integration.
- It features a simple Java client API.
- Data replication between clusters is offered.

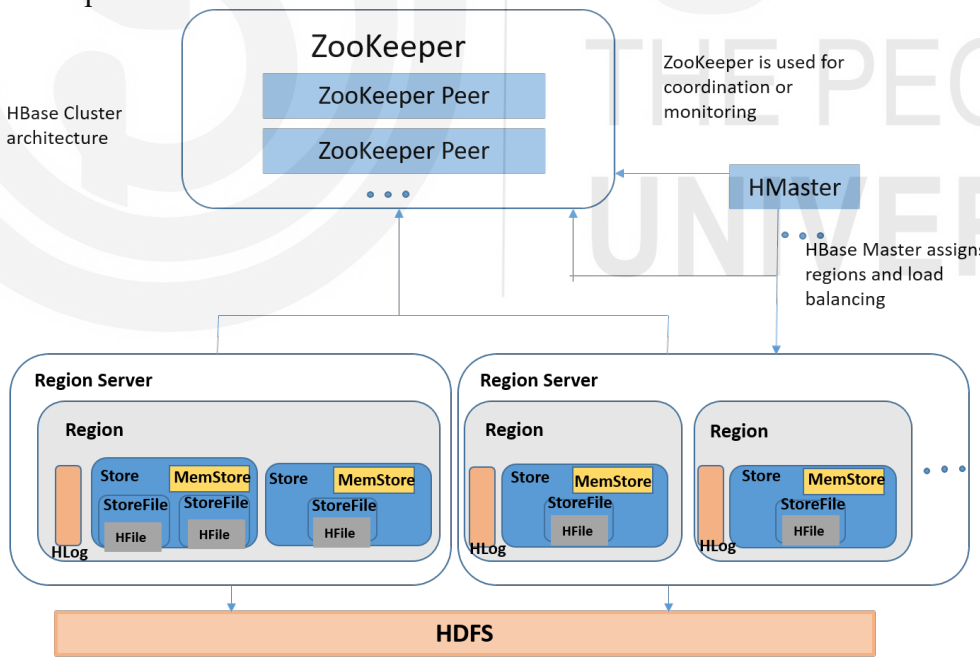


Figure 5: HBase components having HBase master and several region servers.

Tables in HBase are divided into regions and are handled by region servers. Regions are vertically organised into "Stores" by column families. In HDFS, stores are saved as files (as shown in Figure 5). The master server will assign the regions to region servers with Apache ZooKeeper's assistance. It manages the servers in the regions' load balancing. It transfers the regions to less busy servers and unloads the busy servers and negotiates load balancing to maintain the cluster's state. Tables that have been divided and distributed throughout the region servers make up regions. The region server deal with client interactions

and data-related tasks. All read and write requests should be handled by the respective regions. It uses the region size thresholds to determine the region's size. The store includes HFiles and memory store. Similar to a cache memory there is memstore. Everything that is entered into the HBase is initially saved here. The data is then transported, stored as blocks in Hfiles, and the memstore is then cleared. All modifications to the data in HBase's file-based storage are tracked by the Write Ahead Log (WAL). The WAL makes sure that the data changes can be replayed in the event that a Region Server fails or becomes unavailable before the MemStore is cleared. An open-source project called Zookeeper offers functions including naming, maintaining configuration data and provides distributed synchronisation, etc. Different region servers are represented by ephemeral nodes in Zookeeper. These nodes are used by master servers to find servers which are unassigned jobs. Server outages and network splinters are tracked by the nodes. Clients use zookeeper to communicate with the region servers. HBase will handle zookeeper in pseudo mode and standalone mode. HBase operates in standalone mode by default. For the purpose of small-scale testing, standalone mode and pseudo-distributed mode are also offered. Distributed mode is suited for a production setting. HBase daemon instances execute in distributed mode across a number of server machines in the cluster. The HBase architecture in shown in Figure 6.

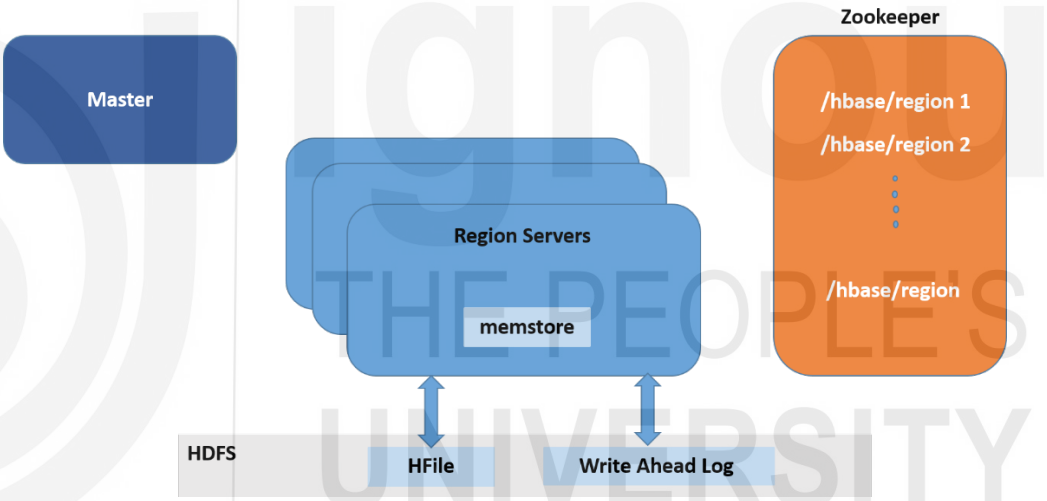


Figure 6: HBase architecture

7.4.1 HBase Installation

The step-by-step installation of HBase is as follows:

We should configure Linux via ssh before installing Hadoop in the environment of Linux (Secure Shell).

Step 1: Check whether JAVA is installed or not, if not you need to install it.

```
$ java -version
```

Step 2: Check whether HADOOP is installed or not, if not you need to install it.

```
$ hadoop version
```

There are three installation options for HBase: standalone, pseudo-distributed, and fully distributed.
Standalone mode:
Step 3: We need to download HBase and then extract tar file:

```
$cd usr/local/
$tar -zxvf hbase-0.98.8-hadoop2-bin.tar.gz
```

Using super user mode move HBase to /usr/local

```
$su
$password: enter your password here
mv hbase-0.99.1/* Hbase/
```

Step 4: We need to configure HBase

```
cd /usr/local/Hbase/conf
gedit hbase-env.sh
```

Replace variable JAVA Home Path with current path:

```
export JAVA_HOME=/usr/lib/jvm/java-1.7.0
```

Edit hbase-site.xml

```
<configuration> //Here you have to set the path where you want HBase to store its files.
<property>
<name>hbase.rootdir</name>
<value>file:/home/hadoop/HBase/HFiles</value>
</property>

//Here you have to set the path where you want HBase to store its built in zookeeper files.
<property>
<name>hbase.zookeeper.property.dataDir</name>
<value>/home/hadoop/zookeeper</value>
</property>

</configuration>
```

Next run HBase start script

```
$cd /usr/local/HBase/bin
$./start-hbase.sh
```

Distributed Mode:

Step 5: Edit hbase-site.xml

```
<property>
   <name>hbase.cluster.distributed</name>
   <value>true</value>
</property>
```

Next, mention the path where HBase is to be run:

```
<property>
   <name>hbase.rootdir</name>
   <value>hdfs://localhost:8030/hbase</value>
</property>
```

Next run HBase start script

```
$cd /usr/local/HBase
$bin/start-hbase.sh
```

Next, check HBase directory in HDFS:

```
$ ./bin/hadoop fs -ls /hbase
```

Step 6: We can start and stop the master:

```
$ ./bin/local-master-backup.sh 2 4
```

```
$ cat /tmp/hbase-user-1-master.pid |xargs kill -9
```

Step 7: We can start and stop the region servers:

```
$ .bin/local-regionservers.sh start 2 3
```

```
$ .bin/local-regionservers.sh stop 3
```

Step 8: We then start HBaseShell

```
$cd $HADOOP_HOME/sbin
$start-all.sh
```

```
$cd /usr/local/HBase
$./bin/start-hbase.sh
```

```
$./bin/local-master-backup.sh start 2 (number signifies specific server.)
```

```
$./bin/./local-regionservers.sh start 3
```

```
$cd bin
$./hbase shell
```

Step 9: Start HBase Web interface:

```
http://localhost:60010
```

After you have completed the installation of HBase, you can use commands to run HBase. Next section discusses these commands.

7.4.2 Working with HBase

In order to interact with HBase, first you should use the shell commands as given below:

HBase shell Commands: One can communicate with HBase by utilising the shell that is included with it. The Hadoop File System is used by HBase to store its data. It contains a master server and region servers. Regions will be used for the data storage (tables). These regions are splited and stored in respective region servers. These region servers are managed by the master server, and HDFS is used for all of these functions. The following commands are supported in HBase shell.

a) Generic Commands:
   status: It gives information about HBase's status, such as how many servers are there.
   version: It gives the HBase current version that is in use.
   table_help: It contains instructions for table related commands.
   whoami: It will provide details of the user.

b) Data definition language:
   create: To create table

list: Lists each table in the HBase database.
Disable: Turns a table into disable mode.
is_disabled: to check if table is disabled
enable: to enable the table
is_enabled: to check if table is enabled
describe: Gives a table's description.
Alter: To alter table
exists : To check if table exists
drop: To drop table
drop_all: Drops all table
Java Admin API: Java offers an Admin API for programmers to implement DDL functionality.

c) Data manipulation language:
Put: Puts a cell value in a specific table at a specific column in a specific row using the put command.
Get: Retrieves a row's or cell's contents.
Delete: Removes a table's cell value.
deleteall: Removes every cell in a specified row.
Scan: Scans the table and outputs the data.
Count: counts the rows in a table and returns that number.
Truncate: A table is disabled, dropped, and then recreated.
Java client API: Prior to all of the aforementioned commands, Java has a client API under the org.apache.hadoop.hbase.client package that enables programmers to perform DML functionality

HBase Programming using Shell

1. Create table:

```
create '<table name>','<column family>'
```

2. List table:

```
hbase(main):001:0 > list
```

This command when used in HBase prompt, gives us the list of all the tables in HBase.

3. Disable table

```
disable 'emp'
```

4. Enable table

```
enable 'emp'
```

5. Describe and alter table

```
hbase> describe 'table name'
```

The syntax to alter a column family's maximum number of cells is provided below.

```
hbase> alter 't1', NAME ⇒ 'f1', VERSIONS ⇒ 5
```

6. Exists table:

```
hbase(main):024:0> exists 'emp'
Table emp does exist

0 row(s) in 0.0750 seconds
```

7. Drop table:

```
hbase(main):019:0> drop 'emp'
0 row(s) in 0.3060 seconds
```

8. Exit shell:

```
hbase(main):021:0> exit
```

9. Insert data:

```
put '<table name>','row1','<colfamily:colname>','<value
```

10. Read data:

```
get '<table name>','row1'
```

11. Delete data:

```
delete '<table name>', '<row>', '<column name >', '<tim
```

12. Scan table:

```
scan '<table name>'
```

13. Count and truncate table:

```
count '<table name>'
```

```
hbase> truncate 'table name'
```

7.5 OTHER TOOLS

A wide variety of Big Data tools and technologies are available on the market today. They improve time management and cost effectiveness for tasks involving data analysis. Some of these include Atlas.ti, HPCC, Apache Storm, Cassandra, StatsIQ, CouchDB, Pentaho, FLink, Cloudera, OpenRefine, RapidMiner etc. Atlas.ti allows to access all available platforms from one place. It can be utilised for mixed techniques and qualitative data analysis varied research. High Performance Computing Cluster (HPCC) Systems provides services using a single platform, architecture, and data processing programming language. Storm is a free large data open source distributed and fault tolerant computing system. Today, a lot of people utilise the Apache Cassandra database to manage massive amounts of data effectively. The statistical tool Stats iQ by Qualtrics is simple to use. JSON documents that can be browsed online or queried using JavaScript are used by CouchDB to store data. Big data technologies are available from Pentaho to extract, prepare, and combine data. It provides analytics and visualisations that transform how any business is run. Apache FLink is open source data analytics tools for massive data stream processing. The most efficient, user-friendly, and highly secure big data platform today is Cloudera. Anyone may access any data from any setting using a single, scalable platform. OpenRefine is a large data analytics tool that aids in working with unclean data, cleaning it up, and converting it between different formats. RapidMiner is utilised for model deployment, machine learning, and data preparation. It provides a range of products to set up predictive analysis and create new data mining methods.

Check Your Progress 3

1. What are the main features of HBase.
2. Explain WAL in HBase.

7.8 SUMMARY

In this unit, we have learnt various tools and cutting-edge big data technologies being used by the data analytics worldwide. Particularly, we studied in detail the usage, installation, components and working of three main big data tools Spark, Hive and HBase. Furthermore, we also discussed how to do query processing in Hive and HBase.

7.9 SOLUTIONS/ANSWERS

Check Your Progress 1

1. The main features of Apache Spark are as follows:

i) Fast processing: In a Hadoop cluster, Spark makes it possible for applications to execute up to 100 times quicker in memory and 10 times faster on storage by reducing the amount of read/write operations,

ii) Multi-language support: Python, Scala, or Java-based built-in APIs are available with Spark. Consequently, you can create applications in a variety of languages. 80 high-level operators are provided by Spark for interactive querying.

iii) Advance Analytics support: Spark offers more than just "Map" and "Reduce." Additionally, it supports graph methods, streaming data, machine learning (ML), and SQL queries.

2. The main components of Apache Spark framework are Spark core, Spark SQL for interactive queries, Spark Streaming for real time streaming analytics, Machine Learning Library, and GraphX for graph processing.

3. Installation Steps for Apache Spark framework:
Step 1: Check whether JAVA is installed or not, if not you need to install it.

```
$ java -version
```

Step 2: Check whether SCALA is installed or not.

```
$scala -version
```

In case it is not installed, you need to download it and
Extract the tar file

```
$ tar xvf scala-2.11.6.tgz
```

We need to move SCALA software files to /usr/ directory as follows:

```
$ su – Password: # cd /home/Hadoop/Downloads/ # mv scala-2.11.6 /usr/local/scala # exit
```

We need to set the environment path variable:

```
$ export PATH = $PATH:/usr/local/scala/bin
```

Last, check if it is installed successfully:

```
$scala -version
```

Step 3: Download Apache Spark
Step 4: Install Spark
We need to extract the requisite tar file

```
$ tar xvf spark-1.3.1-bin-hadoop2.6.tgz
```

We need to move SPARK software files to /usr/ directory as follows:

```
$ su – Password: # cd /home/Hadoop/Downloads/ # mv spark-1.3.1-bin-hadoop2.6 /usr/local/spark # exit
```

We need to set the environment path variable:

```
export PATH=$PATH:/usr/local/spark/bin
```

```
$ source ~/.bashrc
```

Last, check if SPARK is installed successfully:

```
$spark-shell
```

It will be installed successfully.

Check Your Progress 2

1. The main components in Hive architecture can be categorized into:

i) Hive Client: Using JDBC, ODBC, and Thrift drivers, Hive enables applications built in any language, including Python, Java, C++, Ruby, and more. Thus, creating a hive client application in a language of one's choice is quite simple. Hive clients can be of three types:
a) Thrift Clients, b) JDBC Clients c) ODBC Clients

ii) Hive services: Hive offers a number of services, like the Hive server2, Beeline, etc., to handle all queries. Hive provides a range of services, including: a) Beeline, b) Hive Server 2, c) Hive Driver, d) Hive Compiler, e) Optimizer, f) Execution engine, g) Metastore, h) HCatalog i) WebHCat.

iii) Processing and Resource Management: Internally, the de facto engine for Hive's query execution is the MapReduce framework. A software framework called MapReduce is used to create programmes that process enormous amounts of data concurrently on vast clusters of commodity hardware. Data is divided into pieces and processed by map-reduce tasks as part of a map-reduce job.

iv) Distributed Storage: Since Hadoop is the foundation of Hive, the distributed storage is handled by the Hadoop Distributed File System.

2. Hive has 4 main data types:
   1. Column Types: Integer, String, Union, Timestamp
   2. Literals: Floating, Decimal
   3. Null Values: All missing values as NULL
   4. Complex Types: Arrays, Maps, Struct

3. There are 4 main types:
1. Join: Join is extremely similar to SQL's Outer Join.
2. FULL OUTER JOIN: The records from the left and right outer tables are combined in a FULL OUTER JOIN.
3. LEFT OUTER JOIN: All rows from the left table are retrieved using the LEFT OUTER JOIN even if there are no matches in the right table.
4. RIGHT OUTER JOIN: In this case as well, even if there are no matches in the left table, all the rows from the right table are retrieved.

Check Your Progress 3

1. The main features of HBase includes:

- HBase scales linearly.
- It supports automatic failure.
- It offers reliable reading and writing.
- It is both a source and a destination of Hadoop integration.
- It features a simple Java client API.
- Data replication between clusters is offered.

2. All modifications to the data in HBase's file-based storage are tracked by the Write Ahead Log (WAL). The WAL makes sure that the data changes can be replayed in the event that a Region Server fails or becomes unavailable before the MemStore is cleared.

7.10 REFERENCES AND FURTHER READINGS

[1] https://www.infoworld.com/article/3236869/what-is-apache-spark-the-big-data-platform-that-crushed-hadoop.html
[2] https://aws.amazon.com/big-data/what-is-spark/
[3] https://spark.apache.org/
[4] https://www.tutorialspoint.com/hive/hive_views_and_indexes.htm
[5] https://hive.apache.org/downloads.html
[6] https://data-flair.training/blogs/apache-hive-architecture/
[7] https://halvadeforspark.readthedocs.io/en/latest/
[8] https://www.tutorialspoint.com/hbase/index.htm
[9] Capriolo, Edward, Dean Wampler, and Jason Rutherglen. *Programming Hive: Data warehouse and query language for Hadoop*. " O'Reilly Media, Inc.", 2012.
[10] Du, Dayong. *Apache Hive Essentials*. Packt Publishing Ltd, 2015.
[11] Karau, H., Konwinski, A., Wendell, P., & Zaharia, M. (2015). *Learning spark: lightning-fast big data analysis*. " O'Reilly Media, Inc.".
[12] George, Lars. *HBase: the definitive guide: random access to your planet-size data*. " O'Reilly Media, Inc.", 2011.