
Godavari College Of Engineering, Jalgaon.



**ACADEMIC YEAR
2021 - 2022**

VII SEMESTER

**LAB MANUAL
BIG DATA ANALYTICS LAB
[BTCOL707]**

FACULTY : PROF. PRASHANT SHIMPI

**DEPARTMENT OF
COMPUTER ENGINEERING**

**Godavari Foundation's
GODAVARI COLLEGE OF ENGINEERING, JALGAON
(NAAC Accredited)**

(An affiliated to Dr. Babasaheb Ambedkar Technological University)



CERTIFICATE

This is to certify that Miss. **Shweta Ravindra Patil** , Roll No: **02** of **L.Y. COMPUTER** class has satisfactorily carried out the practical work in the Subject : **Big Data Analytics Laboratory [BTCOL707]** as per laid down in the syllabus, in this Laboratory and that this journal represent **her** bonafide work in the year **2021-2022**.

Date :

Signature
PROF. PRASHANT SHIMPI
Faculty in Charge

Signature
PROF. PRAMOD GOSAVI
H.O.D.

Dr. V. H. PATIL
PRINCIPAL
Godavari Foundation's
Godavari College of Engineering, Jalgaon

Index

Sr. No	Practical Name	Page No.	Dates
1	Perform setting up and Installing Hadoop in its two operating modes: a) Pseudo distributed & b) Fully distributed.		
2	Implement the following file management tasks in Hadoop: a) Adding files and directories b) Retrieving files c) Deleting files		
3	To understand the overall programming architecture using Map Reduce API		
4	Store the basic information about students such as roll no, name, date of birth and address Of student using various collection types such as List, Set and Map		
5	Run a basic Word Count Map Reduce program to understand Map Reduce Paradigm.		
6	Install and Run Hbase then use HbaseDDL and DML commands		
7	Install, Deploy & configure Apache Spark Cluster. Run apache spark applications using Scala.		
8	Basic CRUD operations in MongoDB		
9	Retrieve various types of documents from .students collection		
10	Data analytics using Apache Spark on Amazon food dataset, find all the pairs of items frequently reviewed together.		

Subject : Big Data Analytics Lab

Subject Teacher : Prof. Prashant Shimpi

Class : L.Y. B.Tech (Comp)

Date :

Practical No : 1

Roll No : 2

Title : Perform setting up and Installing Hadoop in its two operating modes:

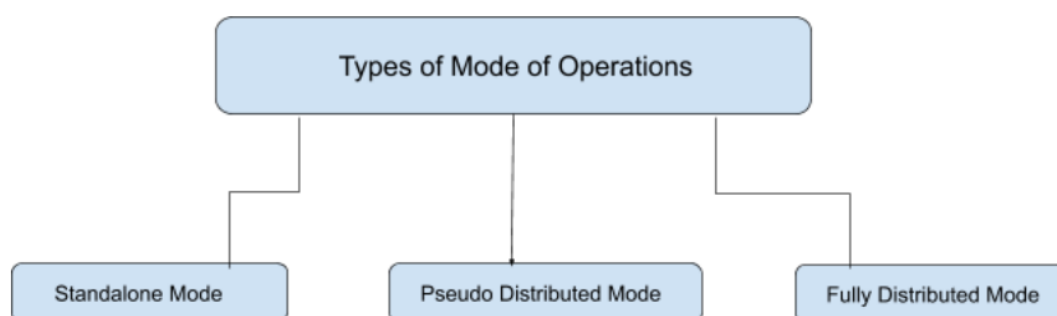
- a) Pseudo distributed.
- b) Fully distributed.

Aim : To Perform setting up and Installing Hadoop in its two operating modes:

- a) Pseudo distributed.
- b) Fully distributed.

Theory :

1 Hadoop : Hadoop is an open-source framework which is mainly used for storage purpose and maintaining and analyzing a large amount of data or datasets on the clusters of commodity hardware, which means it is actually a data management tool. Hadoop also possesses a scale-out storage property, which means that we can scale up or scale down the number of nodes as per the requirement in the future which is really a cool feature.



Hadoop is written in Java, so you will need to have Java installed on your machine, version 6 or later. Sun's JDK is the one most widely used with Hadoop, although others have been reported to work.

Hadoop runs on Unix and on Windows. Linux is the only supported production platform, but other flavors of Unix (including Mac OS X) can be used to run Hadoop for development. Windows is only supported as a development platform, and additionally requires Cygwin to run. During the Cygwin installation process, you should include the openssh package if you plan to run Hadoop in pseudo-distributed mode.

Hadoop Mainly works on 3 different Modes:

- 1) Standalone Mode
- 2) Pseudo-distributed Mode
- 3) Fully-Distributed Mode

1. Standalone Mode : In Standalone Mode none of the Daemon will run i.e. Namenode, Datanode, Secondary Name node, Job Tracker, and Task Tracker. We use job-tracker and task-tracker for processing purposes in Hadoop1. For Hadoop2 we use Resource Manager and Node Manager. Standalone Mode also means that we are installing Hadoop only in a single system. By default, Hadoop is made to run in this Standalone Mode or we can also call it as the Local mode. We mainly use Hadoop in this Mode for the Purpose of Learning, testing, and debugging.

Hadoop works very much Fastest in this mode among all of these 3 modes. As we all know HDFS (Hadoop distributed file system) is one of the major components for Hadoop which utilized for storage Permission is not utilized in this mode. You can think of HDFS as similar to the file system's available for windows i.e. NTFS (New Technology File System) and FAT32(File Allocation Table which stores the data in the blocks of 32 bits). when your Hadoop works in this mode there is no need to configure the files – hdfs-site.xml, mapred-site.xml, core-site.xml for Hadoop environment. In this Mode, all of your Processes will run on a single JVM(Java Virtual Machine) and this mode can only be used for small development purposes.

2. Pseudo Distributed Mode (Single Node Cluster) : In Pseudo-distributed Mode we also use only a single node, but the main thing is that the cluster is simulated, which means that all the processes inside the cluster will run independently to each other. All the daemons that are Namenode, Datanode, Secondary Name node, Resource Manager, Node Manager, etc. will be running as a separate process on separate JVM(Java Virtual Machine) or we can say run on different java processes that is why it is called a Pseudo-distributed.

One thing we should remember that as we are using only the single node set up so all the Master and Slave processes are handled by the single system. Namenode and Resource Manager are used as Master and Datanode and Node Manager is used as a slave. A secondary name node is also used as a Master. The purpose of the Secondary Name node is to just keep the hourly based backup of the Name node. In this Mode,

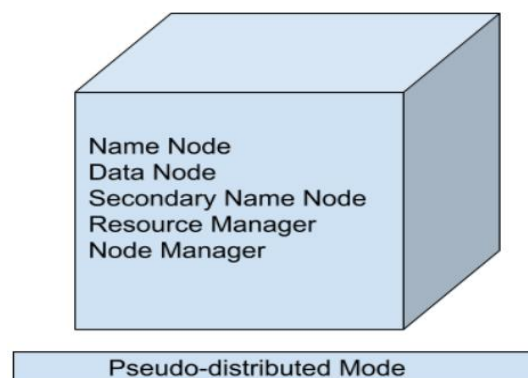


Fig. Pseudo Distributed Mode (Single Node Cluster)

- Hadoop is used for development and for debugging purposes both.
- Our HDFS(Hadoop Distributed File System) is utilized for managing the Input and Output processes.
- We need to change the configuration files mapred-site.xml, core-site.xml, hdfs-site.xml for setting up the environment.

3. Fully Distributed Mode (Multi-Node Cluster) : This is the most important one in which multiple nodes are used few of them run the Master Daemon's that are Namenode and Resource Manager and the rest of them run the Slave Daemon's that are DataNode and Node Manager. Here Hadoop will run on the clusters of Machine or nodes. Here the data that is used is distributed across different nodes. This is actually the Production Mode of Hadoop let's clarify or understand this Mode in a better way in Physical Terminology.

Once you download the Hadoop in a tar file format or zip file format then you install it in your system and you run all the processes in a single system but here in the fully distributed mode we are extracting this tar or zip file to each of the nodes in the Hadoop cluster and then we are using a particular node for a particular process. Once you distribute the process among the nodes then you'll define which nodes are working as a master or which one of them is working as a slave.

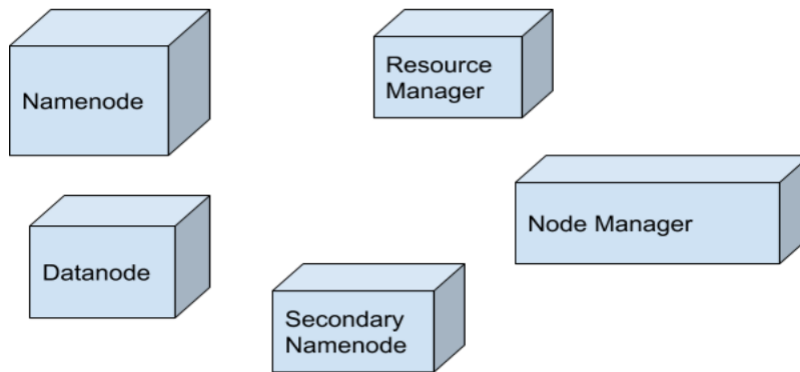


Fig. Fully Distributed Mode (Multi-Node Cluster)

2. Algorithm :

1) Steps Involved in installing Hadoop in Standalone Mode :

1. Command for installing ssh is "**sudo apt-get install ssh**".
2. Command for key generation is **ssh-keygen -t rsa -P ""**.
3. Store the key into rsa.pub by using the command **cat \$HOME/.ssh/id_rsa.pub >> \$HOME/.ssh/authorized_keys**
4. Extract the java by using the command **tar xvfz jdk-8u60-linux-i586.tar.gz**.
5. Extract the eclipse by using the command **tar xvfz eclipse-jee-mars-R-linux-gtk.tar.gz**
6. Extract the hadoop by using the command **tar xvfz hadoop-2.7.1.tar.gz**
7. Move the java to /usr/lib/jvm/ and eclipse to /opt/ paths. Configure the java path in the **eclipse.ini** file
8. Export java path and hadoop path in **./bashrc**
9. Check the installation successful or not by checking the java version and hadoop version
10. Check the hadoop instance in standalone mode working correctly or not by using an implicit hadoop jar file named as word count.
11. If the word count is displayed correctly in **part-r-00000** file it means that standalone mode is installed successfully.

2) Steps Involved in installing Hadoop in Pseudo distributed Mode :

1. In order install pseudo distributed mode we need to configure the hadoop configuration files resides in the directory **/home/lendi/hadoop-2.7.1/etc/hadoop**.
2. First configure the **hadoop-env.sh** file by changing the java path.
3. Configure the **core-site.xml** which contains a property tag, it contains name and

value. Name as **fs.defaultFS** and value as **hdfs://localhost:9000**

4. Configure **hdfs-site.xml**.
5. Configure **yarn-site.xml**.
6. Configure **mapred-site.xml** before configure the copy **mapred-site.xml.template** to **mapred-site.xml**.
7. Now format the name node by using command **hdfs namenode -format**.
8. Type the command **start-dfs.sh** , **start-yarn.sh** means that starts the daemons like NameNode, DataNode, SecondaryNameNode , ResourceManager, NodeManager.
9. Run JPS which views all daemons. Create a directory in the hadoop by using command **hdfs dfs -mkdir /csedir** and enter some data into **lendi.txt** using command **nano lendi.txt** and copy from local directory to hadoop using command **hdfs dfs -copyFromLocal lendi.txt /csedir/** and run sample jar file wordcount to check whether pseudo distributed mode is working or not.
10. Display the contents of file by using command **hdfs dfs -cat /newdir/part-r-00000**.

3) Steps Involved in installing Hadoop in Fully distributed Mode :

1. Stop all single node clusters : **\$stop-all.sh**
2. Decide one as NameNode (Master) and remaining as DataNodes(Slaves).
3. Copy public key to all three hosts to get a password less SSH access
\$ssh-copy-id -I \$HOME/.ssh/id_rsa.pub lendi@l5sys24
4. Configure all Configuration files, to name Master and Slave Nodes.
\$cd \$HADOOP_HOME/etc/hadoop
\$nano core-site.xml
\$ nano hdfs-site.xml
5. Add hostnames to file slaves and save it. : **\$ nano slaves**
6. Configure **\$ nano yarn-site.xml**
7. Do in Master Node
\$ hdfs namenode -format
\$ start-dfs.sh
\$start-yarn.sh
8. Format NameNode
9. Daemons Starting in Master and Slave Nodes
10. END

Input : ubuntu @localhost> jps

Output : Data node, name node Secondary name node, NodeManager, Resource Manager

Conclusion : In this Practical I learn to Perform setting up and Installing Hadoop in its two operating modes:

- a) Pseudo distributed.
- b) Fully distributed.

Subject : Big Data Analytics Lab

Subject Teacher : Prof. Prashant Shimpi

Class : L.Y. B.Tech (Comp)

Date :

Practical No : 2

Roll No : 2

Title : Implement the following file management tasks in Hadoop:

- a) Adding files and directories
- b) Retrieving files
- c) Deleting files

Aim : To Implement the following file management tasks in Hadoop:

- a) Adding files and directories
- b) Retrieving files
- c) Deleting files

Theory :

Hadoop : Hadoop is open-source Project. it runs applications using the MapReduce algorithm, where the data is processed in parallel with others. In short, Hadoop is used to develop applications that could perform complete statistical analysis on huge amounts of data.

Hadoop is an Apache open source framework written in java that allows distributed processing of large datasets across clusters of computers using simple programming models. The Hadoop framework application works in an environment that provides distributed storage and computation across clusters of computers. Hadoop is designed to scale up from single server to thousands of machines, each offering local computation and storage.

Hadoop Architecture : At its core, Hadoop has two major layers namely

- 1. Processing/Computation layer (MapReduce), &
- 2. Storage layer (Hadoop Distributed File System).

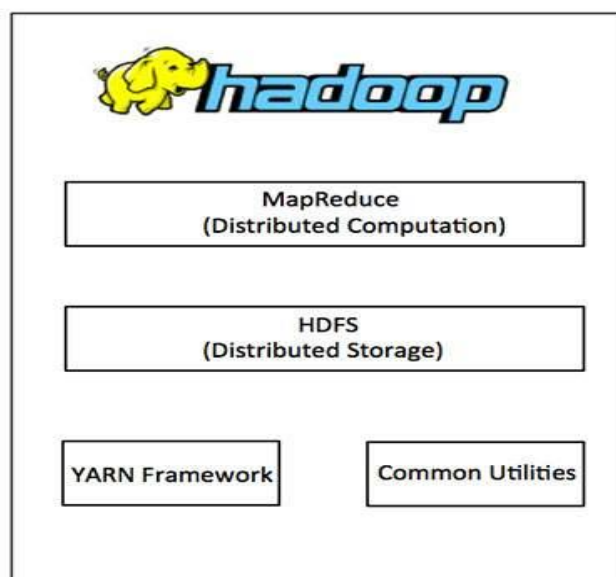


Fig. Hadoop Architecture

1) MapReduce : MapReduce is a parallel programming model for writing distributed applications devised at Google for efficient processing of large amounts of data (multi-terabyte data-sets), on large clusters (thousands of nodes) of commodity hardware in a reliable, fault-tolerant manner. The MapReduce program runs on Hadoop which is an Apache open-source framework.

2) Hadoop Distributed File System (HDFS) :

The Hadoop Distributed File System (HDFS) is based on the Google File System (GFS) and provides a distributed file system that is designed to run on commodity hardware. It has many similarities with existing distributed file systems. However, the differences from other distributed file systems are significant. It is highly fault-tolerant and is designed to be deployed on low-cost hardware. It provides high throughput access to application data and is suitable for applications having large datasets.

HDFS is a scalable distributed filesystem designed to scale to petabytes of data while running on top of the underlying filesystem of the operating system. HDFS keeps track of where the data resides in a network by associating the name of its rack (or network switch) with the dataset. This allows Hadoop to efficiently schedule tasks to those nodes that contain data, or which are nearest to it, optimizing bandwidth utilization. Hadoop provides a set of command line utilities that work similarly to the Linux file commands, and serve as your primary interface with HDFS. We're going to have a look into HDFS by interacting with it from the command line. We will take a look at the most common file management tasks in Hadoop, which include:

1. □ Adding files and directories to HDFS
2. □ Retrieving files from HDFS to local filesystem
3. □ Deleting files from HDFS

Features of HDFS

- It is suitable for the distributed storage and processing.
- Hadoop provides a command interface to interact with HDFS.
- The built-in servers of namenode and datanode help users to easily check the status of cluster.
- Streaming access to file system data.
- HDFS provides file permissions and authentication.

HDFS Architecture : Given below is the architecture of a Hadoop File System.

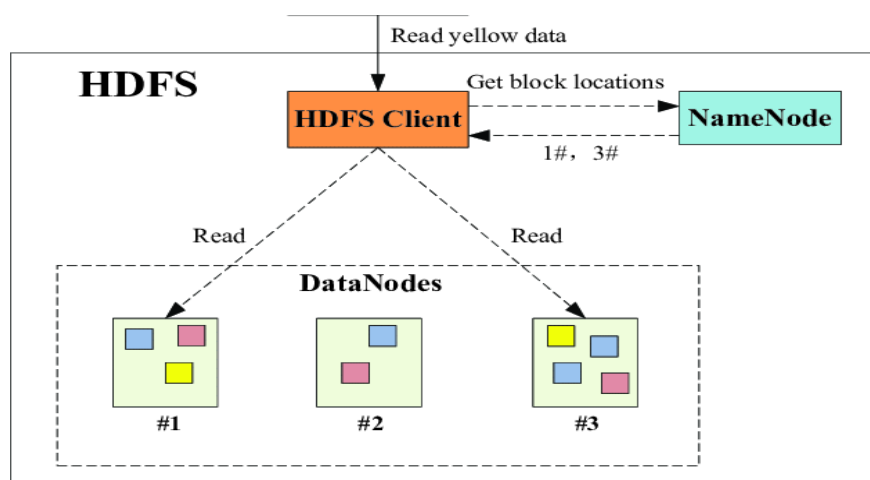


Fig. HDFS Architecture

HDFS follows the master-slave architecture and it has the following elements.

Namenode : The namenode is the commodity hardware that contains the GNU/Linux operating system and the namenode software. It is a software that can be run on commodity hardware. The system having the namenode acts as the master server and it does the following tasks –

- a) Manages the file system namespace.
- b) Regulates client's access to files.
- c) It also executes file system operations such as renaming, closing, and opening files and directories.

Datanode : The datanode is a commodity hardware having the GNU/Linux operating system and datanode software. For every node (Commodity hardware/System) in a cluster, there will be a datanode. These nodes manage the data storage of their system.

- a) Datanodes perform read-write operations on the file systems, as per client request.
- b) They also perform operations such as block creation, deletion, and replication according to the instructions of the namenode.

Block : Generally the user data is stored in the files of HDFS. The file in a file system will be divided into one or more segments and/or stored in individual data nodes. These file segments are called as blocks. In other words, the minimum amount of data that HDFS can read or write is called a Block. The default block size is 64MB, but it can be increased as per the need to change in HDFS configuration.

Goals of HDFS :

- 1. Fault detection and recovery :** Since HDFS includes a large number of commodity hardware, failure of components is frequent. Therefore HDFS should have mechanisms for quick and automatic fault detection and recovery.
- 2. Huge datasets :** HDFS should have hundreds of nodes per cluster to manage the applications having huge datasets.
- 3. Hardware at data :** A requested task can be done efficiently, when the computation takes place near the data. Especially where huge datasets are involved, it reduces the network traffic and increases the throughput.

Apart from the above-mentioned two core components, Hadoop framework also includes the following two modules

3) Hadoop YARN (Yet Another Resource Negotiator) : This is a framework for job scheduling and cluster resource management. YARN is a Framework on which MapReduce works. YARN performs 2 operations that are Job scheduling and Resource Management. The Purpose of Job scheduler is to divide a big task into small jobs so that each job can be assigned to various slaves in a Hadoop cluster and Processing can be Maximized. Job Scheduler also keeps track of which job is important, which job has more priority, dependencies between the jobs and all the other information like job timing, etc. And the use of Resource Manager is to manage all the resources that are made available for running a Hadoop cluster.

Features of YARN :

- Multi-Tenancy
- Scalability
- Cluster-Utilization
- Compatibility

4. Hadoop common or Common Utilities : Hadoop common or Common utilities are nothing but our java library and java files or we can say the java scripts that we need for all the other components present in a Hadoop cluster. these utilities are used by HDFS, YARN, and MapReduce for running the cluster. Hadoop Common verify that Hardware failure in a Hadoop cluster is common so it needs to be solved automatically in software by Hadoop Framework.

These are Java libraries and utilities required by other Hadoop modules.

Task 1 : Adding files and directories to HDFS : Before you can run Hadoop programs on data stored in HDFS, you'll need to put the data into HDFS first. Let's create a directory and put a file in it. HDFS has a default working directory of /user/\$USER, where \$USER is your login user name. This directory isn't automatically created for you, though, so let's create it with the mkdir command. For the purpose of illustration, we use chuck. You should substitute your user name in the example commands.

```
hadoop fs -mkdir /user/chuck
hadoop fs -put example.txt
hadoop fs -put example.txt /user/chuck
```

Task 2 : Retrieving files from HDFS to local filesystem : The Hadoop command get copies files from HDFS back to the local filesystem. To retrieve example.txt, we can run the following command:

```
hadoop fs -cat example.txt
```

Task 3 : Deleting files from HDFS :

```
hadoop fs -rm example.txt
```

- Command for creating a directory in hdfs is "hdfs dfs -mkdir /lendicse".
- Adding directory is done through the command "hdfs dfs -put lendi_english /"

Conclusion : In this Practical I learn to Implement the following file management tasks in Hadoop:

- a) Adding files and directories
- b) Retrieving files
- c) Deleting files

Subject : Big Data Analytics Lab

Subject Teacher : Prof. Prashant Shimpi

Class : L.Y. B.Tech (Comp)

Date :

Practical No : 3

Roll No : 2

Title : Understand the overall programming architecture using Map Reduce API

Aim : To understand the overall programming architecture using Map Reduce API

Theory :

Big Data : Big Data can be termed as that colossal load of data that can be hardly processed using the traditional data processing units. A better example of Big Data would be the currently trending Social Media sites like Facebook, Instagram, WhatsApp and YouTube.

Hadoop : Hadoop is a Big Data framework designed and deployed by Apache Foundation. It is an open-source software utility that works in the network of computers in parallel to find solutions to Big Data and process it using the MapReduce algorithm.

Google released a paper on MapReduce technology in December 2004. This became the genesis of the Hadoop Processing Model. So, MapReduce is a programming model that allows us to perform parallel and distributed processing on huge data sets.

Traditional Approach : In this approach, an enterprise will have a computer to store and process big data. For storage purpose, the programmers will take the help of their choice of database vendors such as Oracle, IBM, etc. In this approach, the user interacts with the application, which in turn handles the part of data storage and analysis.

This approach works fine with those applications that process less voluminous data that can be accommodated by standard database servers, or up to the limit of the processor that is processing the data. But when it comes to dealing with huge amounts of scalable data, it is a hectic task to process such data through a single database bottleneck.

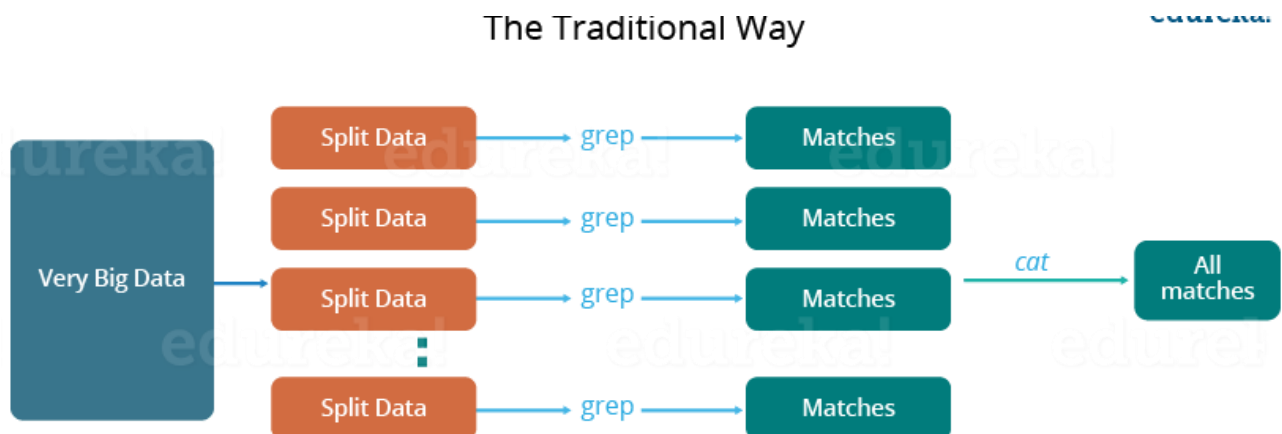


Fig. Traditional Approach

Challenges with Traditional Approach : Let us look at the challenges associated with this traditional approach:

- **Critical path problem:** It is the amount of time taken to finish the job without delaying the next milestone or actual completion date. So, if, any of the machines delay the job, the whole work gets delayed.
- **Reliability problem:** What if, any of the machines which are working with a part of data fails? The management of this failover becomes a challenge.
- **Equal split issue:** How will I divide the data into smaller chunks so that each machine gets even part of data to work with. In other words, how to equally divide the data such that no individual machine is overloaded or underutilized.
- **The single split may fail:** If any of the machines fail to provide the output, I will not be able to calculate the result. So, there should be a mechanism to ensure this fault tolerance capability of the system.
- **Aggregation of the result:** There should be a mechanism to aggregate the result generated by each of the machines to produce the final output.

To overcome these issues, we have the MapReduce framework which allows us to perform such parallel computations without bothering about the issues like reliability, fault tolerance etc. Therefore, MapReduce gives you the flexibility to write code logic without caring about the design issues of the system.

MapReduce : MapReduce is a programming model used for efficient processing in parallel over large data-sets in a distributed manner. The data is first split and then combined to produce the final result. The libraries for MapReduce is written in so many programming languages with various different-different optimizations.

The purpose of MapReduce in Hadoop is to Map each of the jobs and then it will reduce it to equivalent tasks for providing less overhead over the cluster network and to reduce the processing power. The MapReduce task is mainly divided into two phases Map Phase and Reduce Phase.

MapReduce Architecture :

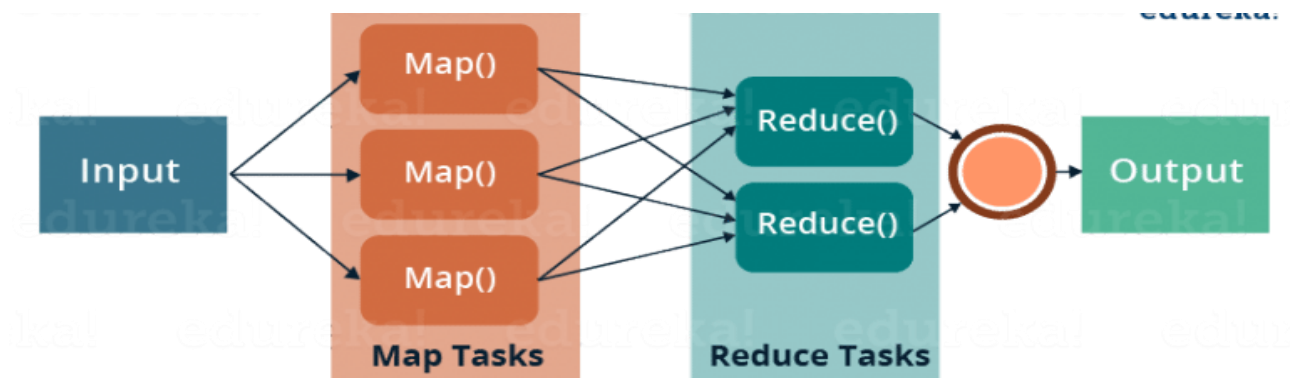


Fig. MapReduce Architecture

Phases of MapReduce in Big Data : Hadoop divides MapReduce input job into tasks. There are two types of tasks:

1. Map tasks (Splits & Mapping)
2. Reduce tasks (Shuffling, Reducing)

a) Input Splits : An input to a MapReduce in Big Data job is divided into fixed-size pieces called input splits. Input split is a chunk of the input that is consumed by a single map.

b) Mapping : This is the very first phase in the execution of map-reduce program. In this phase, data in each split is passed to a mapping function to produce output values. In our example, a job of mapping phase is to count a number of occurrences of each word from input splits (more details about input-split is given below) and prepare a list in the form of <word, frequency>

c) Shuffling : This phase consumes the output of Mapping phase. Its task is to consolidate the relevant records from Mapping phase output. In our example, the same words are clubbed together along with their respective frequency.

d) Reducing : In this phase, output values from the Shuffling phase are aggregated. This phase combines values from Shuffling phase and returns a single output value. In short, this phase summarizes the complete dataset.

The complete execution process (execution of Map and Reduce tasks, both) is controlled by two types of entities called :

1) Jobtracker : Acts like a master (responsible for complete execution of submitted job)

2) Multiple Task Trackers : Acts like slaves, each of them performing the job

For every job submitted for execution in the system, there is one Jobtracker that resides on Namenode and there are multiple tasktrackers which reside on Datanode.

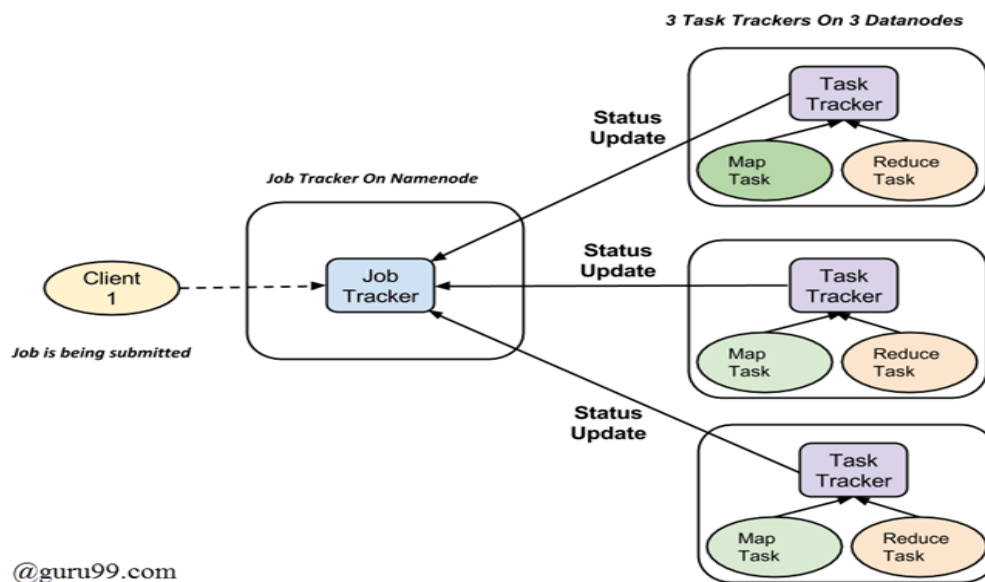


Fig. Working Of Hadoop MapReduce

Conclusion : In this Practical I learn to understand the overall programming architecture using Map Reduce API

Subject : Big Data Analytics Lab

Subject Teacher : Prof. Prashant Shimpi

Class : L.Y. B.Tech (Comp)

Date :

Practical No : 4

Roll No : 2

Title : Store the basic information about students such as roll no, name, date of birth and address Of student using various collection types such as List, Set and Map

Aim : To Store the basic information about students such as roll no, name, date of birth and address Of student using various collection types such as List, Set and Map

Theory :

Collection in Java : Any group of individual objects which are represented as a single unit is known as the collection of the objects. In Java, a separate framework named the “Collection Framework” has been defined in JDK 1.2 which holds all the collection classes and interface in it.

The Collection interface (java.util.Collection) and Map interface (java.util.Map) are the two main “root” interfaces of Java collection classes.

Collections class is a member of the Java Collections Framework. The java.util.Collections package is the package that contains the Collections class. Collections class is basically used with the static methods that operate on the collections or return the collection. All the methods of this class throw the NullPointerException if the collection or object passed to the methods is null.

Framework: A framework is a set of classes and interfaces which provide a ready-made architecture. In order to implement a new feature or a class, there is no need to define a framework. However, an optimal object-oriented design always includes a framework with a collection of classes such that all the classes perform the same kind of task.

Need of Java collection : There are several benefits of using Java collections such as:

- Reducing the effort required to write the code by providing useful data structures and algorithms
- Java collections provide high-performance and high-quality data structures and algorithms thereby increasing the speed and quality
- Unrelated APIs can pass collection interfaces back and forth
- Decreases extra effort required to learn, use, and design new API's
- Supports reusability of standard data structures and algorithms

Java Collection Framework Hierarchy : As we have learned Java collection framework includes interfaces and classes. Now, let us see the Java collections framework hierarchy.

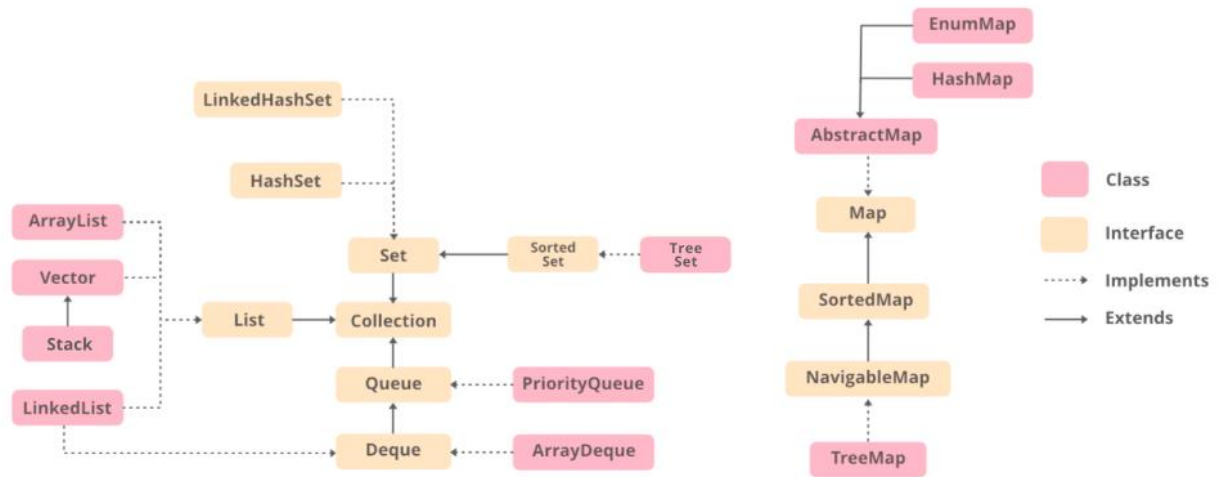


Fig. Java Collection Framework

Code :

```
// Java Program to illustrate the
// addition of elements in a List

import java.util.*;
public class GFG {

    public static void main(String args[])
    {

        // Creating a List
        List<String> student = new ArrayList<>();

        // Adding elements in the List
        student.add("student 1");
        student.add("student 2");
        student.add("student 3");

        // Iterating the List
        // element using for-each loop
        for (String child : student)
            System.out.println(child);
    }
}
```

Output : student 1
student 2
student 3

References : <https://www.edureka.co/blog/java-collections/>

Conclusion : In this Practical I learn to Store the basic information about students such as roll no, name, date of birth and address Of student using various collection types such as List, Set and Map.

Subject : Big Data Analytics Lab

Subject Teacher : Prof. Prashant Shimpi

Class : L.Y. B.Tech (Comp)

Date :

Practical No : 5

Roll No : 2

Title : Run a basic Word Count Map Reduce program to understand Map Reduce Paradigm.

- a) Find the number of occurrence of each word appearing in the input file(s)
- b) Performing a MapReduce Job for word search count (look for specific keywords in a file)

Aim : To Run a basic Word Count Map Reduce program to understand Map Reduce Paradigm.

Theory :

MapReduce : MapReduce is a programming framework that allows us to perform distributed and parallel processing on large data sets in a distributed environment.

- MapReduce consists of two distinct tasks : Map and Reduce.
- As the name MapReduce suggests, the reducer phase takes place after the mapper phase has been completed.
- So, the first is the map job, where a block of data is read and processed to produce key-value pairs as intermediate outputs.
- The output of a Mapper or map job (key-value pairs) is input to the Reducer.
- The reducer receives the key-value pair from multiple map jobs.
- Then, the reducer aggregates those intermediate data tuples (intermediate key-value pair) into a smaller set of tuples or key-value pairs which is the final output.
- Let us understand more about MapReduce and its components. MapReduce majorly has the following three Classes. They are,
 - **1) Mapper Class :** The first stage in Data Processing using MapReduce is the Mapper Class. Here, RecordReader processes each Input record and generates the respective key-value pair. Hadoop's Mapper store saves this intermediate data into the local disk.
 - **Input Split :** It is the logical representation of data. It represents a block of work that contains a single map task in the MapReduce Program.
 - **RecordReader :** It interacts with the Input split and converts the obtained data in the form of Key-Value Pairs.
 - **2) Reducer Class :** The Intermediate output generated from the mapper is fed to the reducer which processes it and generates the final output which is then saved in the HDFS.
 - **3) Driver Class :** The major component in a MapReduce job is a Driver Class. It is responsible for setting up a MapReduce Job to run-in Hadoop. We specify the names of Mapper and Reducer Classes long with data types and their respective job names.

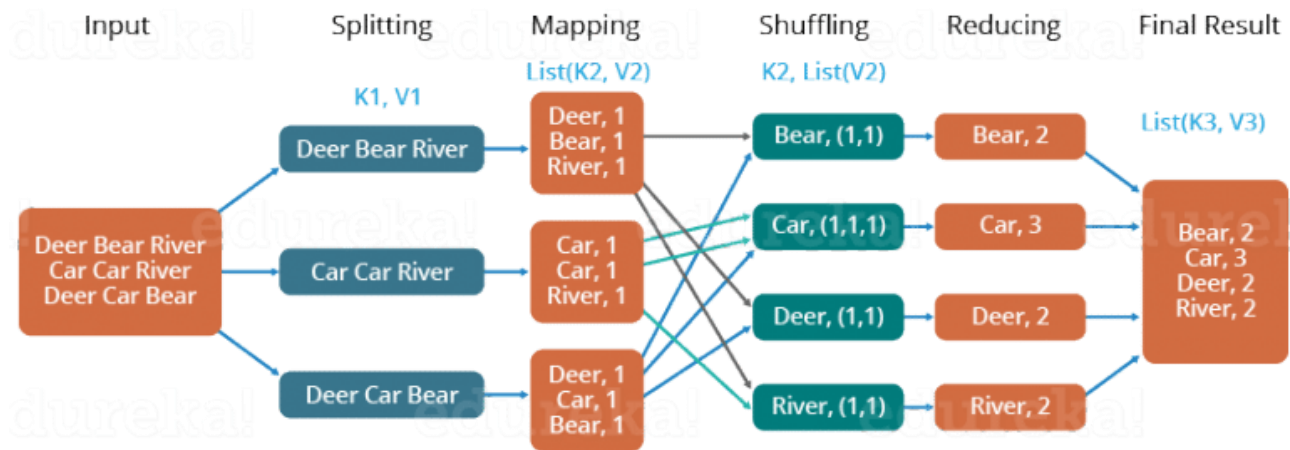
A Word Count Example of MapReduce : Let us understand, how a MapReduce works by taking an example where I have a text file called example.txt whose contents are as follows:

Deer, Bear, River, Car, Car, River, Deer, Car and Bear

Now, suppose, we have to perform a word count on the sample.txt using MapReduce. So, we will be finding the unique words and the number of occurrences of those unique words.

The Overall MapReduce Word Count Process

edureka!



Word Counting Process :

1. First, we divide the input into three splits as shown in the figure. This will distribute the work among all the map nodes.
2. Then, we tokenize the words in each of the mappers and give a hardcoded value (1) to each of the tokens or words. The rationale behind giving a hardcoded value equal to 1 is that every word, in itself, will occur once.
3. Now, a list of key-value pair will be created where the key is nothing but the individual words and value is one. So, for the first line (Deer Bear River) we have 3 key-value pairs – Deer, 1; Bear, 1; River, 1. The mapping process remains the same on all the nodes.
4. After the mapper phase, a partition process takes place where sorting and shuffling happen so that all the tuples with the same key are sent to the corresponding reducer.
5. So, after the sorting and shuffling phase, each reducer will have a unique key and a list of values corresponding to that very key. For example, Bear, [1,1]; Car, [1,1,1].., etc.
6. Now, each Reducer counts the values which are present in that list of values. As shown in the figure, reducer gets a list of values which is [1,1] for the key Bear. Then, it counts the number of ones in the very list and gives the final output as – Bear, 2.
7. Finally, all the output key/value pairs are then collected and written in the output file.

Code :

1) Mapper Code:

```
// Importing libraries
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.Mapper;
```

```

import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reporter;

public class WCMapper extends MapReduceBase implements Mapper<LongWritable,
                                                                    Text, Text, IntWritable> {

    // Map function
    public void map(LongWritable key, Text value, OutputCollector<Text,
                                                                    IntWritable> output, Reporter rep) throws IOException
    {

        String line = value.toString();

        // Splitting the line on spaces
        for (String word : line.split(" "))
        {
            if (word.length() > 0)
            {
                output.collect(new Text(word), new IntWritable(1));
            }
        }
    }
}

```

2) Reducer Code:

```

// Importing libraries
import java.io.IOException;
import java.util.Iterator;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reducer;
import org.apache.hadoop.mapred.Reporter;

public class WCReducer extends MapReduceBase implements Reducer<Text,
                                                                    IntWritable, Text, IntWritable> {

    // Reduce function
    public void reduce(Text key, Iterator<IntWritable> value,
                                                                    OutputCollector<Text, IntWritable> output,
                                                                    Reporter rep) throws IOException
    {

        int count = 0;

        // Counting the frequency of each words
        while (value.hasNext())
        {

```

```

        IntWritable i = value.next();
        count += i.get();
    }

    output.collect(key, new IntWritable(count));
}
}

```

3) Driver Code:

```

// Importing libraries
import java.io.IOException;
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.FileInputFormat;
import org.apache.hadoop.mapred.FileOutputFormat;
import org.apache.hadoop.mapred.JobClient;
import org.apache.hadoop.mapred.JobConf;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;

public class WCDriver extends Configured implements Tool {

    public int run(String args[]) throws IOException
    {
        if (args.length < 2)
        {
            System.out.println("Please give valid inputs");
            return -1;
        }

        JobConf conf = new JobConf(WCDriver.class);
        FileInputFormat.setInputPaths(conf, new Path(args[0]));
        FileOutputFormat.setOutputPath(conf, new Path(args[1]));
        conf.setMapperClass(WCMapper.class);
        conf.setReducerClass(WCReducer.class);
        conf.setMapOutputKeyClass(Text.class);
        conf.setMapOutputValueClass(IntWritable.class);
        conf.setOutputKeyClass(Text.class);
        conf.setOutputValueClass(IntWritable.class);
        JobClient.runJob(conf);
        return 0;
    }

    // Main Method
    public static void main(String args[]) throws Exception
    {
        int exitCode = ToolRunner.run(new WCDriver(), args);
    }
}

```

```
        System.out.println(exitCode);  
    }  
}
```

Input: Hello I am Student
 Hello I am an Intern

Output: Hello : 2
 I : 2
 am : 2
 an : 1
 Intern : 1
 Student : 1

Conclusion : In this Practical I learn to Run a basic Word Count Map Reduce program to understand Map Reduce Paradigm.

- a) Find the number of occurrence of each word appearing in the input file(s)
- b) Performing a MapReduce Job for word search count (look for specific keywords in a file)

Subject : Big Data Analytics Lab

Subject Teacher : Prof. Prashant Shimpi

Class : L.Y. B.Tech (Comp)

Date :

Practical No : 6

Roll No : 2

Title : Install and Run Hbase then use HbaseDDL and DML commands

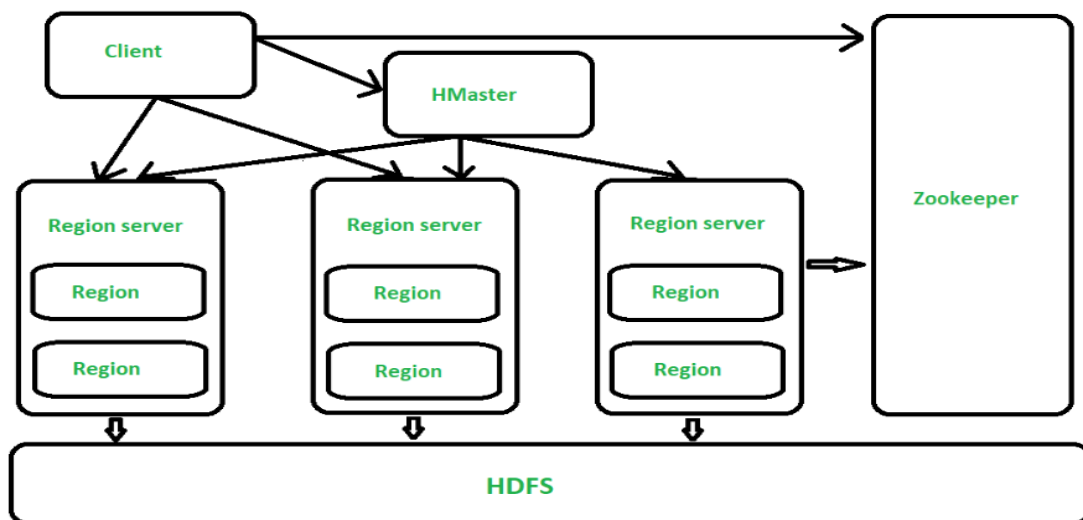
Aim : To Install and Run Hbase then use HbaseDDL and DML commands

Theory :

Hbase : Hbase is an open source and sorted map data built on Hadoop. It is column oriented and horizontally scalable.

It is based on Google's Big Table. It has set of tables which keep data in key value format. Hbase is well suited for sparse data sets which are very common in big data use cases. Hbase provides APIs enabling development in practically any programming language. It is a part of the Hadoop ecosystem that provides random real-time read/write access to data in the Hadoop File System.

Hbase Architecture : HBase architecture has 3 main components: HMaster, Region Server, Zookeeper.



All the 3 components are described below:

1. **HMaster :** The implementation of Master Server in HBase is HMaster. It is a process in which regions are assigned to region server as well as DDL (create, delete table) operations. It monitor all Region Server instances present in the cluster. In a distributed environment, Master runs several background threads. HMaster has many features like controlling load balancing, failover etc.

2. **Region Server** : HBase Tables are divided horizontally by row key range into Regions. Regions are the basic building elements of HBase cluster that consists of the distribution of tables and are comprised of Column families. Region Server runs on HDFS DataNode which is present in Hadoop cluster. Regions of Region Server are responsible for several things, like handling, managing, executing as well as reads and writes HBase operations on that set of regions. The default size of a region is 256 MB.
3. **Zookeeper** : It is like a coordinator in HBase. It provides services like maintaining configuration information, naming, providing distributed synchronization, server failure notification etc. Clients communicate with region servers via zookeeper.

Advantages of HBase :

- Can store large data sets
- Database can be shared
- Cost-effective from gigabytes to petabytes
- High availability through failover and replication

Disadvantages of Hbase :

- No support SQL structure
- No transaction support
- Sorted only on key
- Memory issues on the cluster

Comparison between HBase and HDFS:

- HBase provides low latency access while HDFS provides high latency operations.
- HBase supports random read and writes while HDFS supports Write once Read Many times.
- HBase is accessed through shell commands, Java API, REST, Avro or Thrift API while HDFS is accessed through MapReduce jobs.

Why Hbase :

- RDBMS get exponentially slow as the data becomes large
- Expects data to be highly structured, i.e. ability to fit in a well-defined schema
- Any change in schema might require a downtime
- For sparse datasets, too much of overhead of maintaining NULL values

Features of Hbase :

1. Horizontally scalable : You can add any number of columns anytime.
2. Automatic Failover : Automatic failover is a resource that allows a system administrator to automatically switch data handling to a standby system in the event of system compromise
3. Integrations with Map/Reduce framework : All the commands and java codes internally implement Map/ Reduce to do the task and it is built over Hadoop Distributed File System.
4. fundamentally, it's a platform for storing and retrieving data with random access.
5. It doesn't care about datatypes(storing an integer in one row and a string in another for the same column).

6. It doesn't enforce relationships within your data.
7. It provides easy to use Java API for client access.
8. Consistency : We can use this HBase feature for high-speed requirements because it offers consistent reads and writes. HBase provides consistent read and writes.
9. Atomic Read and Write : During one read or write process, all other processes are prevented from performing any read or write operations this is what we call Atomic read and write. So, HBase offers atomic read and write, on a row level.
10. Sharding : In order to reduce I/O time and overhead, HBase offers automatic and manual splitting of regions into smaller subregions, as soon as it reaches a threshold size.
11. High Availability : Moreover, it offers LAN and WAN which supports failover and recovery. Basically, there is a master server, at the core, which handles monitoring the region servers as well as all metadata for the cluster.
12. Client API : Through Java APIs, it also offers programmatic access.
13. Scalability : In both linear and modular form, HBase supports scalability. In addition, we can say it is linearly scalable.
14. Hadoop/HDFS integration : HBase can run on top of other file systems as well as like Hadoop/HDFS integration.
15. Distributed storage : This feature of HBase supports distributed storage such as HDFS.
16. Data Replication : HBase supports data replication across clusters.
17. Failover Support and Load Sharing : By using multiple block allocation and replications, HDFS is internally distributed and automatically recovered and HBase runs on top of HDFS, hence HBase is automatically recovered. Also using RegionServer replication, this failover is facilitated.
18. MapReduce Support : For parallel processing of large volume of data, HBase supports MapReduce.
19. Backup Support : In HBase "Backup support" means it supports back-up of Hadoop MapReduce jobs in HBase tables.
20. Sorted Row Keys : It is possible to build an optimized request Since searching is done on the range of rows, and HBase stores row keys in lexicographical orders, hence, by using these sorted row keys and timestamp we can build an optimized request.
21. Real-time Processing : In order to perform real-time query processing, HBase supports block cache and Bloom filters.
22. Faster Lookups : While it comes to faster lookups, HBase internally uses Hash tables and offers random access, as well as it stores the data in indexed HDFS files.
23. Type of Data : For both semi-structured as well as structured data, HBase supports well.
24. Schema-less : There is no concept of fixed columns schema in HBase because it is schema-less. Hence, it defines only column families.
25. High Throughput : Due to high security and easy management characteristics of HBase, it offers unprecedented high write throughput.

HBase Installation :

1. The prerequisite for HBase installation are Java and Hadoop installed on your Linux machine.
2. Hbase can be installed in three modes: standalone, Pseudo Distributed mode and Fully Distributed mode.
3. Download the Hbase package from <http://www.interior-dsgn.com/apache/hbase/stable/> and unzip it with the below commands.

```
$cd usr/local/$wget http://www.interior-dsgn.com/apache/hbase/stable/hbase-0.98.8-hadoop2-bin.tar.gz
$tar -zxvf hbase-0.98.8-hadoop2-bin.tar.gz
```


4. Login as super user as shown below

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

5. Configuring HBase in Standalone Mode : Set the java Home for HBase and open hbase-env.sh file from the conf folder. Edit JAVA_HOME environment variable and change the existing path to your current JAVA_HOME variable as shown below.

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

6. Replace the existing JAVA_HOME value with your current value as shown below.

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

7. Inside /usr/local/Hbase you will find hbase-site.xml. Open it and within configuration add the below code.

```
<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>
```

8. Now start the Hbase by running the start-hbase.sh present in the bin folder of Hbase.

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

9. Cloudera VM is recommended as it has Hbase preinstalled on it. Starting Hbase: Type Hbase shell in terminal to start the hbase.

HBase Shell Commands : HBase Shell Commands can be categorized into below types.

1. HBase Shell General Commands
2. Data Definition Commands
3. Data Manipulation Commands

4. Other HBase Shell Commands

1. General Commands

1. status – shows the cluster status
2. table_help – help on Table reference commands, scan, put, get, disable, drop etc.
3. version – displays HBase version
4. whoami – shows the current HBase user.

2. DDL Commands :

1. alter : add/modify/delete column families, as well as change table configuration
2. create : Used for Creating tables. Pass a table name, and a set of column family specifications (at least one), and, optionally, table configuration as arguments.
3. describe : Prints the schema of a table. We can also use abbreviated 'desc' for the same thing.
4. disable : disable an existing HBase table. Disabled tables will not be deleted from HBase but they are not available for regular access. This table is excluded from the list command and we can not run any other command except either enable or drop commands on disabled tables. Disabling is similar to deleting the tables temporarily.
5. disable_all : Disable all of tables matching the given regex:
6. drop : Dropping of HBase tables means deleting the tables permanently. To drop a table it must first be disabled.
7. drop_all : Drop all of the tables matching the given regex:
8. enable : Used to enable a table which might be currently disabled.
9. enable_all : Enable all of the tables matching the given regex:
10. exists : To check the existence of an HBase Table
11. get_table : Gets the given table name and return it as an actual object to be manipulated by the user.
12. is_disabled : To know whether an HBase table is disabled or not.
13. is_enabled : To know whether an HBase table is enabled or not.
14. list : List all tables in hbase. Optional regular expression parameter could be used to filter the output. Examples:
15. show_filters : Show all the filters in hbase. Example:

3) DML Command :

1. **append** : Appends a cell 'value' at specified table/row/column coordinates.
2. **count** : Count the number of rows in a table.
3. **delete** : Put a delete cell value at specified table/row/column and optionally timestamp coordinates. Deletes must match the deleted cell's coordinates exactly.
4. **deleteall** : Delete all cells in a given row; pass a table name, row, and optionally a column and timestamp. Examples:
5. **get** : Get row or cell contents; pass table name, row, and optionally a dictionary of column(s), timestamp, timerange and versions. Examples:
6. **get_counter** : Return a counter cell value at specified table/row/column coordinates.
7. **incr** : Increments a cell 'value' at specified table/row/column coordinates. To increment a cell value in table 'ns1:t1' or 't1' at row 'r1' under column 'c1' by 1 (can be omitted) or 10 do:
8. **put** : Put a cell 'value' at specified table/row/column and optionally timestamp coordinates. To put a cell value into table 'ns1:t1' or 't1' at row 'r1' under column 'c1' marked with the time 'ts1', do:
9. **truncate** : Disables, drops and recreates the specified table. After truncate of an HBase table, schema will be present but not the records.
10. **truncate_preserve** : Disables, drops and recreates the specified table while still maintaining the previous region boundaries.
11. **scan** : This command Scan a table; pass table name and optionally a dictionary of scanner specifications. Scanner specifications may include one or more of: TIMERANGE, FILTER, LIMIT, STARTROW, STOPROW, TIMESTAMP, MAXLENGTH, or COLUMNS, CACHE
 - If no columns are specified, all columns will be scanned. To scan all members of a column family, leave the qualifier empty as in 'col_family:'.
 - The filter can be specified in two ways:
 1. Using a filterString
 2. Using the entire package name of the filter.

References : <http://hadooptutorial.info/hbase-shell-commands-in-practice/>

Conclusion : In this Practical I learn to Install and Run Hbase then use HbaseDDL and DML commands

Subject : Big Data Analytics Lab

Subject Teacher : Prof. Prashant Shimpi

Class : L.Y. B.Tech (Comp)

Date :

Practical No : 7

Roll No : 2

Title : Install, Deploy & configure Apache Spark Cluster. Run apache spark applications using Scala.

Aim : To Install, Deploy & configure Apache Spark Cluster. Run apache spark applications using Scala.

Theory :

Apache Spark : Apache Spark is an open-source, distributed processing system used for big data workloads. It utilizes in-memory caching, and optimized query execution for fast analytic queries against data of any size. It provides development APIs in Java, Scala, Python and R, and supports code reuse across multiple workloads—batch processing, interactive queries, real-time analytics, machine learning, and graph processing. You'll find it used by organizations from any industry, including at FINRA, Yelp, Zillow, DataXu, Urban Institute, and CrowdStrike. Apache Spark has become one of the most popular big data distributed processing framework with 365,000 meetup members in 2017.

Apache Spark Architecture : Apache Spark works in a master-slave architecture where the master is called “Driver” and slaves are called “Workers”. When you run a Spark application, Spark Driver creates a context that is an entry point to your application, and all operations (transformations and actions) are executed on worker nodes, and the resources are managed by Cluster Manager.

A spark cluster has a single Master and any number of Slaves/Workers. The driver and the executors run their individual Java processes and users can run them on the same horizontal spark cluster or on separate machines.

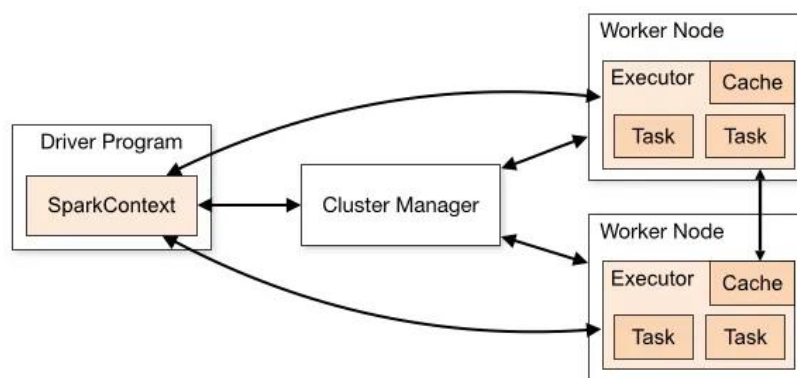


Fig. Architecture of Apache Spark.

Following are most important takeaways of the architecture:

- Each application gets its own executor processes, which remains in memory up to the duration of the complete application and run tasks in multiple threads. This means each application is independent from the other, on both the scheduling side since each driver schedules its own tasks and executor side as tasks from different applications run in different JVMs.
- Spark is independent of cluster managers that implies, it can be coupled with any cluster manager and then leverage that cluster.
- Because the driver schedules tasks on the cluster, it should be run as close to the worker nodes as possible.

Cluster Manager Types : As of writing this Apache Spark Tutorial, Spark supports below cluster managers:

1. **Standalone** : a simple cluster manager included with Spark that makes it easy to set up a cluster.
2. **Apache Mesos** : Mesos is a Cluster manager that can also run Hadoop MapReduce and Spark applications.
3. **Hadoop YARN** : the resource manager in Hadoop 2. This is mostly used, cluster manager.
4. **Kubernetes** : an open-source system for automating deployment, scaling, and management of containerized applications.

Modes of Apache Spark Deployment : let's understand how we can deploy spark to our systems :

1. **Standalone Mode in Apache Spark** : Spark is deployed on the top of Hadoop Distributed File System (HDFS). For computations, Spark and MapReduce run in parallel for the Spark jobs submitted to the cluster.
2. **Hadoop YARN/ Mesos** : Apache Spark runs on Mesos or YARN (Yet another Resource Navigator, one of the key features in the second-generation Hadoop) without any root-access or pre-installation. It integrates Spark on top Hadoop stack that is already present on the system.
3. **SIMR (Spark in Map Reduce)** : This is an add-on to the standalone deployment where Spark jobs can be launched by the user and they can use the spark shell without any administrative access.

Spark Modules or Components of Spark :

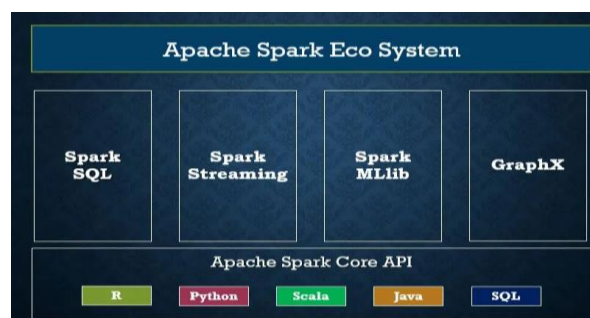


Fig. Apache Spark Modules

1. **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.
2. **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.
3. **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.
4. **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).
5. **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.

Apache Spark Features :

- In-memory computation
- Distributed processing using parallelize
- Can be used with many cluster managers (Spark, Yarn, Mesos e.t.c)
- Fault-tolerant
- Immutable
- Cache & persistence
- Inbuild-optimization when using DataFrames
- Supports ANSI SQL

Apache Spark Advantages :

- Spark is a general-purpose, in-memory, fault-tolerant, distributed processing engine that allows you to process data efficiently in a distributed fashion.
- Applications running on Spark are 100x faster than traditional systems.
- You will get great benefits using Spark for data ingestion pipelines.
- Using Spark we can process data from Hadoop HDFS, AWS S3, Databricks DBFS, Azure Blob Storage, and many file systems.
- Spark also is used to process real-time data using Streaming and Kafka.
- Using Spark Streaming you can also stream files from the file system and also stream from the socket.
- Spark natively has machine learning and graph libraries.

Apache Spark Cluster Installation and Configuration Guide :

Step 1 : Pre-requisite setups :

Each node in our cluster (one master + three workers) should have latest Java installed. We will use apt to install Java. We will first update that.

```
$ sudo apt update
$ sudo apt install openjdk-8-jre-headless
$ java --version
```

Step 2 : We also need to install Scala in all of our machines.

```
$ sudo apt install scala
$ scala -version
```

Step 3 : Setting up Keyless SSH : In order to allow the communication between spark, we need to setup Keyless SSH login between master and the workers. This allows masters to coordinate running of different executors. If your master is not able to communicate with your workers, kindly check the security groups and your keyless SSH setup to resolve them.

```
// Step 3.1 --> Install openssh-server and openssh-client on the master only.
$ sudo apt install openssh-server openssh-client

// Now generate key pairs:
$ ssh-keygen -t rsa -P ""

//Use the following command in order to make this key an authorized one:
cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys

//Now we need to copy the content of .ssh/id_rsa.pub (of master) to .ssh/authorized_keys (of all
the slaves as well as master). Use these commands:
ssh-copy-id user@pd-master
ssh-copy-id user@pd-slave1
ssh-copy-id user@pd-slave2

// Let's check if everything went well, try to connect to the slaves:
$ ssh slave01
$ ssh slave02

// As you can see everything went well, to exit just type the command: :: $exit
```

Step 4: Installing Spark :

```
// Now we Download the latest version of Apache Spark.
$ wget http://www-us.apache.org/dist/spark/spark-2.4.4/spark-2.4.4-bin-hadoop2.7.tgz

// Extract the Apache Spark file you just downloaded
$ tar xvf spark-2.4.4-bin-hadoop2.7.tgz

// Move Apache Spark software files : Use the following command to move the spark software
files to respective directory (/usr/local/bin)
$ sudo mv spark-2.4.4-bin-hadoop2.7 /usr/local/spark

// Set up the environment for Apache Spark : Edit the bashrc file using this command:
$ sudo gedit ~/.bashrc

// Add the following line to the file. This adds the location where the spark software file are
located to the PATH variable.
export PATH = $PATH:/usr/local/spark/bin

// Now we need to use the following command for sourcing the ~/.bashrc file:
$ source ~/.bashrc
```

Step 5: Configuring Master to keep track of its workers : Now that we have Spark installed in all our machines we need to let the Master instance know about the different workers that is available. We will be using a standalone cluster manager for demonstration purposes.

```
//Apache Spark Master Configuration (do this step on the Master VM only)
//Edit spark-env.sh Move to spark conf folder and create a copy of the template of spark-env.sh and
rename it.
$ cd /usr/local/spark/conf
$ cp spark-env.sh.template spark-env.sh

// Now edit the configuration file spark-env.sh.
$ sudo vim spark-env.sh

// And add the following parameters:
export SPARK_MASTER_HOST='<MASTER-IP>'export
JAVA_HOME=<Path_of_JAVA_installation>

// Add Workers --> Edit the configuration file slaves in (/usr/local/spark/conf).
$ sudo nano slaves

// And add the following entries.
pd-master
pd-slave01
pd-slave02
```

Step 6 : Let's try to start our Apache Spark Cluster :

```
// To start the spark cluster, run the following command on master.:
$ cd /usr/local/spark
$ ./sbin/start-all.sh

// I won't stop it, but in case you want to stop the cluster, this is the command:
$ ./sbin/stop-all.sh

//To check if the services started we use the command:
$ jps
```

Definition: Cluster Manager : is an agent that works in allocating the resource requested by the master on all the workers. The cluster manager then shares the resource back to the master, which the master assigns to a particular driver program.

```
// 5.1 --> We need to modify /usr/local/spark/conf/spark-env.sh file by providing information
about Java location and the master node's IP. This will be added to the environment when running
Spark jobs.

# contents of conf/spark-env.sh
export SPARK_MASTER_HOST=<master-private-ip>
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
# For PySpark use
export PYSARK_PYTHON=python3
```



```
// 5.2 --> We will also add all the IPs where the worker will be started. Open the
/usr/local/spark/conf/slaves file and paste the following.
# contents of conf/slaves
<worker-private-ip1>
<worker-private-ip2>
<worker-private-ip3>
```

Code : Here is an example application code that generates 4 million random alphanumeric string with length 5 and persists them into outputDir.

```
/* GenerateNames.scala */
import org.apache.spark.SparkContext
import org.apache.spark.SparkConf
import scala.util.Random
object GenerateNames {
  val outputDir = "/home/jung/sparkapp/output/part"
  def main(args: Array[String]) {
    val conf = new SparkConf()
    .setMaster("local[3]")
    .setAppName("GenerateNames")
    val sc = new SparkContext(conf)
    for (partition <- 0 to 3) {
      val data = Seq.fill(1000000)(Random.alphanumeric.take(5).mkString)
      sc.parallelize(data, 1).saveAsTextFile(outputDir + "_" + partition)
    }
  }
}
```

References :

- 1) <https://www.projectpro.io/apache-spark-tutorial/spark-tutorial>
- 2) https://courses.cs.duke.edu/fall15/compsci290.1/TA_Material/jungkang/how_to_run_spark_app.pdf
- 3) <https://sparkbyexamples.com/>
- 4) <https://sparkbyexamples.com/spark/spark-setup-on-hadoop-yarn/>
- 5) <https://codeflex.co/apache-spark-cluster-installation-and-configuration-guide/>
- 6) <https://blog.insightdatascience.com/simply-install-spark-cluster-mode-341843a52b88>
- 7) https://medium.com/@jootorres_11979/how-to-install-and-set-up-an-apache-spark-cluster-on-hadoop-18-04-b4d70650ed42

Conclusion : In this Practical I learn to Install, Deploy & configure Apache Spark Cluster. Run apache spark applications using Scala.

Subject : Big Data Analytics Lab

Subject Teacher : Prof. Prashant Shimpi

Class : L.Y. B.Tech (Comp)

Date :

Practical No : 8

Roll No : 2

Title : Basic CRUD operations in MongoDB

Aim : To Perform Basic CRUD operations in MongoDB

Theory :

MongoDB : MongoDB, the most popular NoSQL database, is an open-source document-oriented database. The term 'NoSQL' means 'non-relational'. It means that MongoDB isn't based on the table-like relational database structure but provides an altogether different mechanism for storage and retrieval of data. This format of storage is called BSON (similar to JSON format).

A simple MongoDB document Structure:

```
{  
  title: 'Geeksforgeeks',  
  by: 'Harshit Gupta',  
  url: 'https://www.geeksforgeeks.org',  
  type: 'NoSQL'  
}
```

SQL databases store data in tabular format. This data is stored in a predefined data model which is not very much flexible for today's real-world highly growing applications. Modern applications are more networked, social and interactive than ever. Applications are storing more and more data and are accessing it at higher rates.

Relational Database Management System(RDBMS) is not the correct choice when it comes to handling big data by the virtue of their design since they are not horizontally scalable. If the database runs on a single server, then it will reach a scaling limit. NoSQL databases are more scalable and provide superior performance. MongoDB is such a NoSQL database that scales by adding more and more servers and increases productivity with its flexible document model.

RDBMS vs MongoDB:

1. RDBMS has a typical schema design that shows number of tables and the relationship between these tables whereas MongoDB is document-oriented. There is no concept of schema or relationship.
2. Complex transactions are not supported in MongoDB because complex join operations are not available.
3. MongoDB allows a highly flexible and scalable document structure. For example, one data document of a collection in MongoDB can have two fields whereas the other document in the same collection can have four.

4. MongoDB is faster as compared to RDBMS due to efficient indexing and storage techniques.
5. There are a few terms that are related in both databases. What's called Table in RDBMS is called a Collection in MongoDB. Similarly, a Tuple is called a Document and A Column is called a Field. MongoDB provides a default '_id' (if not provided explicitly) which is a 12-byte hexadecimal number that assures the uniqueness of every document. It is similar to the Primary key in RDBMS.

MongoDB	RDBMS
It is a non-relational and document-oriented database.	It is a relational database.
It is suitable for hierarchical data storage.	It is not suitable for hierarchical data storage.
It has a dynamic schema.	It has a predefined schema.
It centers around the CAP theorem (Consistency, Availability, and Partition tolerance).	It centers around ACID properties (Atomicity, Consistency, Isolation, and Durability).
In terms of performance, it is much faster than RDBMS.	In terms of performance, it is slower than MongoDB.

Terminologies:

1. **Container** : A MongoDB Database can be called as the container for all the collections.
2. **Collection** : Collection is a bunch of MongoDB documents. It is similar to tables in RDBMS.
3. **Document** : Document is made of fields. It is similar to a tuple in RDBMS, but it has dynamic schema here. Documents of the same collection need not have the same set of fields

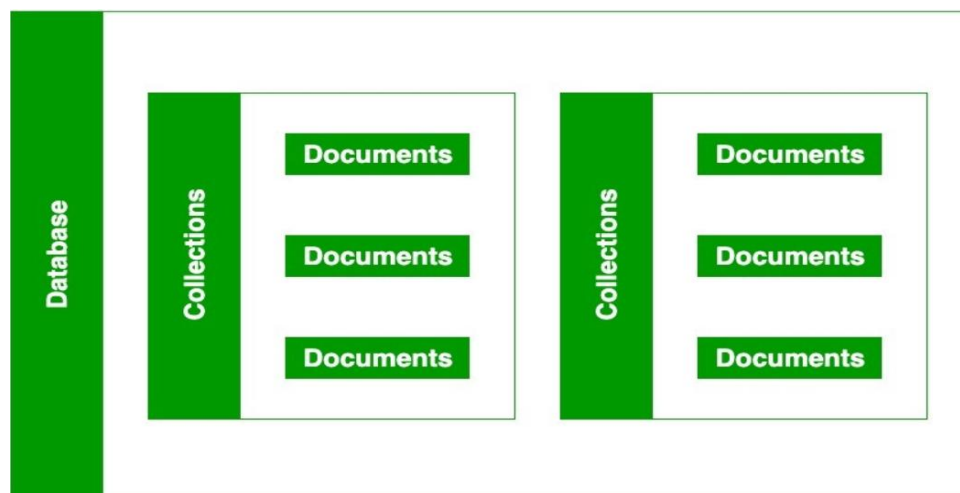


Fig. MongoDB Storage Architecture.

MongoDB CRUD Operation : MongoDB provides a set of some basic but most essential

operations that will help you to easily interact with the MongoDB server and these operations are known as CRUD operations.

1. **Create Operations :** The create or insert operations are used to insert or add new documents in the collection. If a collection does not exist, then it will create a new collection in the database. You can perform, create operations using the following methods provided by the MongoDB:

db.collection.insertOne()	It is used to insert a single document in the collection.
db.collection.insertMany()	It is used to insert multiple documents in the collection.

Example :

```
anki — mongo — 80x55
[> use GeeksforGeeks
switched to db GeeksforGeeks
> db.student.insertOne({
... name : "Sumit",
... age : 20,
... branch : "CSE",
... course : "C++ STL",
... mode : "online",
... paid : true,
... amount : 1499
[... ]})
{
  "acknowledged" : true,
  "insertedId" : ObjectId("5e540cdc92e6dfa3fc48ddae")
}
>
```

2. **Read Operations :** The Read operations are used to retrieve documents from the collection, or in other words, read operations are used to query a collection for a document. You can perform read operation using the following method provided by the MongoDB:

db.collection.find()	It is used to retrieve documents from the collection.
-----------------------------	---

Example :

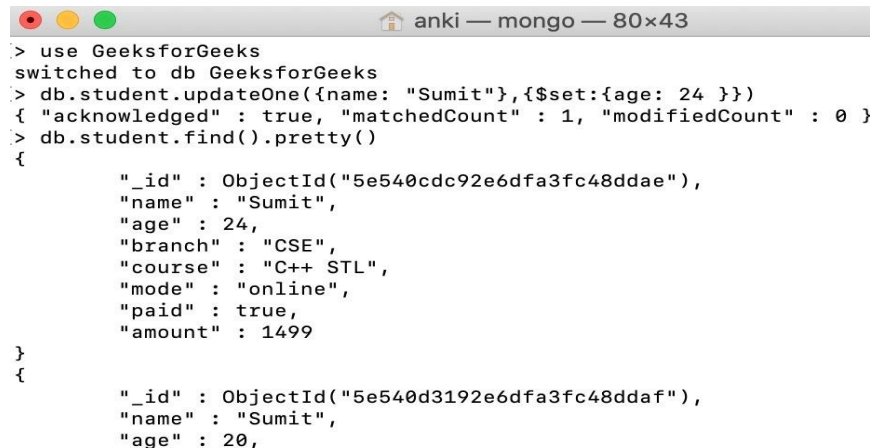
```
anki — mongo — 80x55
[> use GeeksforGeeks
switched to db GeeksforGeeks
[> db.student.find().pretty()
{
  "_id" : ObjectId("5e540cdc92e6dfa3fc48ddae"),
  "name" : "Sumit",
  "age" : 20,
  "branch" : "CSE",
  "course" : "C++ STL",
  "mode" : "online",
  "paid" : true,
  "amount" : 1499
}
{
  "_id" : ObjectId("5e540d3192e6dfa3fc48ddaf"),
  "name" : "Sumit",
  "age" : 20,
  "branch" : "CSE",
  "course" : "C++ STL",
  "mode" : "online",
  "paid" : true,
  "amount" : 1499
}
r
```

3. **Update Operations :** The update operations are used to update or modify the existing

document in the collection. You can perform update operations using the following methods provided by the MongoDB:

db.collection.updateOne()	It is used to update a single document in the collection that satisfy the given criteria.
db.collection.updateMany()	It is used to update multiple documents in the collection that satisfy the given criteria.
db.collection.replaceOne()	It is used to replace single document in the collection that satisfy the given criteria.

Example :



```

> use GeeksforGeeks
switched to db GeeksforGeeks
> db.student.updateOne({name: "Sumit"},{$set:{age: 24 }})
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 0 }
> db.student.find().pretty()
{
  "_id" : ObjectId("5e540cdc92e6dfa3fc48ddae"),
  "name" : "Sumit",
  "age" : 24,
  "branch" : "CSE",
  "course" : "C++ STL",
  "mode" : "online",
  "paid" : true,
  "amount" : 1499
}
{
  "_id" : ObjectId("5e540d3192e6dfa3fc48ddaf"),
  "name" : "Sumit",
  "age" : 20,

```

4. **Delete Operations :** The delete operation are used to delete or remove the documents from a collection. You can perform delete operations using the following methods provided by the MongoDB:

db.collection.deleteOne()	It is used to delete a single document from the collection that satisfy the given criteria.
db.collection.deleteMany()	It is used to delete multiple documents from the collection that satisfy the given criteria.

Example :

```

}
> db.student.deleteOne({name: "Sumit"})
{ "acknowledged" : true, "deletedCount" : 1 }
> db.student.find().pretty()
{
  "_id" : ObjectId("5e540d3192e6dfa3fc48ddaf"),
  "name" : "Sumit",
  "age" : 20,
  "branch" : "CSE",
  "course" : "C++ STL",
  "mode" : "online",
  "paid" : true,
  "amount" : 1499,
  "year" : 2020
}
{
  "_id" : ObjectId("5e54103592e6dfa3fc48ddb1"),
  "name" : "Rohit",
  "age" : 21,
  "branch" : "CSE",
  "course" : "C++ STL",
  "mode" : "online",
  "paid" : true,
  "amount" : 1499
}

```

Features of MongoDB :

1. **Schema-less Database:** It is the great feature provided by the MongoDB. A Schema-less database means one collection can hold different types of documents in it. Or in other words, in the MongoDB database, a single collection can hold multiple documents and these documents may consist of the different numbers of fields, content, and size. It is not necessary that the one document is similar to another document like in the relational databases. Due to this cool feature, MongoDB provides great flexibility to databases.
2. **Document Oriented:** In MongoDB, all the data stored in the documents instead of tables like in RDBMS. In these documents, the data is stored in fields(key-value pair) instead of rows and columns which make the data much more flexible in comparison to RDBMS. And each document contains its unique object id.
3. **Indexing:** In MongoDB database, every field in the documents is indexed with primary and secondary indices this makes easier and takes less time to get or search data from the pool of the data. If the data is not indexed, then database search each document with the specified query which takes lots of time and not so efficient.
4. **Scalability:** MongoDB provides horizontal scalability with the help of sharding. Sharding means to distribute data on multiple servers, here a large amount of data is partitioned into data chunks using the shard key, and these data chunks are evenly distributed across shards that reside across many physical servers. It will also add new machines to a running database.
5. **Replication:** MongoDB provides high availability and redundancy with the help of replication, it creates multiple copies of the data and sends these copies to a different server so that if one server fails, then the data is retrieved from another server.
6. **Aggregation:** It allows to perform operations on the grouped data and get a single result or computed result. It is similar to the SQL GROUPBY clause. It provides three different aggregations i.e, aggregation pipeline, map-reduce function, and single-purpose aggregation methods
7. **High Performance:** The performance of MongoDB is very high and data persistence as compared to another database due to its features like scalability, indexing, replication, etc.

Advantages of MongoDB :

- It is a schema-less NoSQL database. You need not to design the schema of the database when you are working with MongoDB.
- It does not support join operation.
- It provides great flexibility to the fields in the documents.
- It contains heterogeneous data.
- It provides high performance, availability, scalability.
- It supports Geospatial efficiently.
- It is a document oriented database and the data is stored in BSON documents.
- It also supports multiple document ACID transition(string from MongoDB 4.0).
- It does not require any SQL injection.
- It is easily integrated with Big Data Hadoop

Disadvantages of MongoDB :

- It uses high memory for data storage.
- You are not allowed to store more than 16MB data in the documents.
- The nesting of data in BSON is also limited you are not allowed to nest data more than 100 levels.

Where do we use MongoDB : MongoDB is preferred over RDBMS in the following scenarios:

1. **Big Data:** If you have huge amount of data to be stored in tables, think of MongoDB before RDBMS databases. MongoDB has built-in solution for partitioning and sharding your database.
2. **Unstable Schema:** Adding a new column in RDBMS is hard whereas MongoDB is schema-less. Adding a new field does not effect old documents and will be very easy.
3. **Distributed data** Since multiple copies of data are stored across different servers, recovery of data is instant and safe even if there is a hardware failure.

Language Support by MongoDB : MongoDB currently provides official driver support for all popular programming languages like C, C++, Rust, C#, Java, Node.js, Perl, PHP, Python, Ruby, Scala, Go, and Erlang.

Conclusion : In this Practical I learn to Perform Basic CRUD operations in MongoDB

Subject : Big Data Analytics Lab

Subject Teacher : Prof. Prashant Shimpi

Class : L.Y. B.Tech (Comp)

Date :

Practical No : 9

Roll No : 2

Title : Retrieve various types of documents from students collection

Aim : To Retrieve various types of documents from students collection

Theory :

Retrieve documents from a collection in MongoDB : The method of fetching or getting data from a MongoDB database is carried out by using MongoDB queries. While performing a query operation, one can also use criteria's or conditions which can be used to retrieve specific data from the database.

MongoDB provides a function called **db.collection.find()** which is used for retrieval of documents from a MongoDB database.

Syntax :

```
db.yourCollectionName.find();
```

The above syntax will return all the documents from a collection in MongoDB. To understand the above syntax, let us create a collection with documents.

The query to create documents are as follows:

```
> db.retrieveAllStudents.insertOne({"StudentId":"STUD101","StudentName":"David","StudentAge":24});
{
  "acknowledged" : true, "insertedId" : ObjectId("5c6bf5cf68174aae23f5ef4e")
}
> db.retrieveAllStudents.insertOne({"StudentId":"STUD102","StudentName":"Carol","StudentAge":22});
{
  "acknowledged" : true, "insertedId" : ObjectId("5c6bf5e968174aae23f5ef4f")
}
> db.retrieveAllStudents.insertOne({"StudentId":"STUD103","StudentName":"Maxwell","StudentAge":25});
{
  "acknowledged" : true, "insertedId" : ObjectId("5c6bf5f768174aae23f5ef50")
}
> db.retrieveAllStudents.insertOne({"StudentId":"STUD104","StudentName":"Bob","StudentAge":23});
{
  "acknowledged" : true, "insertedId" : ObjectId("5c6bf60868174aae23f5ef51")
}
> db.retrieveAllStudents.insertOne({"StudentId":"STUD105","StudentName":"Sam","StudentAge":27});
{
  "acknowledged" : true, "insertedId" : ObjectId("5c6bf61b68174aae23f5ef52")
}
```


Retrieve various types of documents from students collection :

1. Now you can use the above syntax in order to retrieve all the documents from a collection with the help of find() method. The query is as follows:

```
> db.retrieveAllStudents.find();
```

Output:

```
{ "_id" : ObjectId("5c6bf5cf68174aae23f5ef4e"), "StudentId" : "STUD-101",  
  "StudentName" :  
    "David", "StudentAge" : 24 }  
{ "_id" : ObjectId("5c6bf5e968174aae23f5ef4f"), "StudentId" : "STUD-102",  
  "StudentName" :  
    "Carol", "StudentAge" : 22 }
```

2. For a proper formatted output, use pretty() with find(). The query is as follows:

```
> db.retriveAllStudents.find().pretty();
```

Output :

```
{  
  "_id" : ObjectId("5c6bf5cf68174aae23f5ef4e"),  
  "StudentId" : "STUD-101",  
  "StudentName" : "David",  
  "StudentAge" : 24  
}  
{  
  "_id" : ObjectId("5c6bf5e968174aae23f5ef4f"),  
  "StudentId" : "STUD-102",  
  "StudentName" : "Carol",  
  "StudentAge" : 22  
}
```

3. If you want to retrieve a single document on the basis of some condition, then you can use the following query. Here, we are retrieving the document with StudentName as "Maxwell":

```
> db.retriveAllStudents.find({"StudentName":"Maxwell"}).pretty();
```

Output :

```
{  
  "_id" : ObjectId("5c6bf5f768174aae23f5ef50"),  
  "StudentId" : "STUD-103",  
  "StudentName" : "Maxwell",  
  "StudentAge" : 25  
}
```

References : <https://www.educba.com/mongodb-list-collections/>

Conclusion : In this Practical I learn to Retrieve various types of documents from students collection

Subject : Big Data Analytics Lab

Subject Teacher : Prof. Prashant Shimpi

Class : L.Y. B.Tech (Comp)

Date :

Practical No : 10

Roll No : 2

Title : Data analytics using Apache Spark on Amazon food dataset, find all the pairs of items frequently reviewed together.

a) Write a single Spark application that:

- i. Transposes the original Amazon food dataset, obtaining a PairRDD of the type: <user_id> → <list of the product_ids reviewed by user_id>
- ii. Counts the frequencies of all the pairs of products reviewed together;
- iii. Writes on the output folder all the pairs of products that appear more than once and their frequencies. The pairs of products must be sorted by frequency.

Aim : To Perform Data analytics using Apache Spark on Amazon food dataset, find all the pairs of items frequently reviewed together.

Theory :

Apache Spark : Apache Spark is an open-source, distributed processing system used for big data workloads. It utilizes in-memory caching, and optimized query execution for fast analytic queries against data of any size. It provides development APIs in Java, Scala, Python and R, and supports code reuse across multiple workloads—batch processing, interactive queries, real-time analytics, machine learning, and graph processing. You'll find it used by organizations from any industry, including at FINRA, Yelp, Zillow, DataXu, Urban Institute, and CrowdStrike. Apache Spark has become one of the most popular big data distributed processing framework with 365,000 meetup members in 2017.

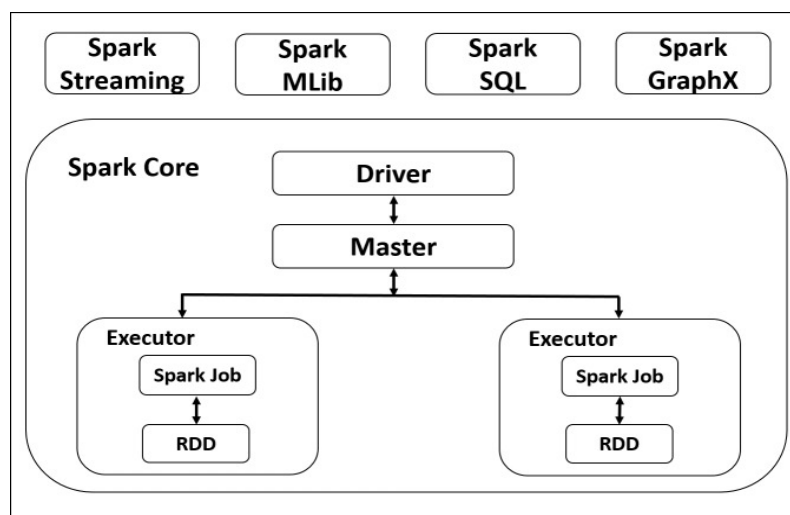


Fig. Apache Spark Architecture

The goal of Spark was to create a new framework, optimized for fast iterative processing like machine learning, and interactive data analysis, while retaining the scalability, and fault tolerance of Hadoop MapReduce. The first paper entitled, “Spark: Cluster Computing with Working Sets” was published in June 2010, and Spark was open sourced under a BSD license. In June, 2013, Spark

entered incubation status at the Apache Software Foundation (ASF), and established as an Apache Top-Level Project in February, 2014. Spark can run standalone, on Apache Mesos, or most frequently on Apache Hadoop.

Apache Spark working : Hadoop MapReduce is a programming model for processing big data sets with a parallel, distributed algorithm. Developers can write massively parallelized operators, without having to worry about work distribution, and fault tolerance. However, a challenge to MapReduce is the sequential multi-step process it takes to run a job. With each step, MapReduce reads data from the cluster, performs operations, and writes the results back to HDFS. Because each step requires a disk read, and write, MapReduce jobs are slower due to the latency of disk I/O.

- Spark was created to address the limitations to MapReduce, by doing processing in-memory, reducing the number of steps in a job, and by reusing data across multiple parallel operations.
- With Spark, only one-step is needed where data is read into memory, operations performed, and the results written back—resulting in a much faster execution.
- Spark also reuses data by using an in-memory cache to greatly speed up machine learning algorithms that repeatedly call a function on the same dataset. Data re-use is accomplished through the creation of DataFrames, an abstraction over Resilient Distributed Dataset (RDD), which is a collection of objects that is cached in memory, and reused in multiple Spark operations. This dramatically lowers the latency making Spark multiple times faster than MapReduce, especially when doing machine learning, and interactive analytics.

Apache Spark vs. Apache Hadoop : Outside of the differences in the design of Spark and Hadoop MapReduce, many organizations have found these big data frameworks to be complimentary, using them together to solve a broader business challenge.

- **Hadoop** is an open source framework that has the Hadoop Distributed File System (HDFS) as storage, YARN as a way of managing computing resources used by different applications, and an implementation of the MapReduce programming model as an execution engine. In a typical Hadoop implementation, different execution engines are also deployed such as Spark, Tez, and Presto.
- **Spark** is an open source framework focused on interactive query, machine learning, and real-time workloads. It does not have its own storage system, but runs analytics on other storage systems like HDFS, or other popular stores like Amazon Redshift, Amazon S3, Couchbase, Cassandra, and others. Spark on Hadoop leverages YARN to share a common cluster and dataset as other Hadoop engines, ensuring consistent levels of service, and response.

Benefits of Apache Spark : There are many benefits of Apache Spark to make it one of the most active projects in the Hadoop ecosystem. These include:

- **Fast :** Through in-memory caching, and optimized query execution, Spark can run fast analytic queries against data of any size.
- **Developer friendly :** Apache Spark natively supports Java, Scala, R, and Python, giving you a variety of languages for building your applications. These APIs make it easy for

your developers, because they hide the complexity of distributed processing behind simple, high-level operators that dramatically lowers the amount of code required.

- **Multiple workloads** : Apache Spark comes with the ability to run multiple workloads, including interactive queries, real-time analytics, machine learning, and graph processing. One application can combine multiple workloads seamlessly.

Apache Spark Workloads : The Spark framework includes:

- Spark Core as the foundation for the platform
- Spark SQL for interactive queries
- Spark Streaming for real-time analytics
- Spark MLlib for machine learning
- Spark GraphX for graph processing

Apache Spark use cases : Spark is a general-purpose distributed processing system used for big data workloads. It has been deployed in every type of big data use case to detect patterns, and provide real-time insight. Example use cases include:

- **Financial Services** : Spark is used in banking to predict customer churn, and recommend new financial products. In investment banking, Spark is used to analyze stock prices to predict future trends.
- **Healthcare** : Spark is used to build comprehensive patient care, by making data available to front-line health workers for every patient interaction. Spark can also be used to predict/recommend patient treatment.
- **Manufacturing** : Spark is used to eliminate downtime of internet-connected equipment, by recommending when to do preventive maintenance.
- **Retail** : Spark is used to attract, and keep customers through personalized services and offers.

Spark and RDDs :

- **1) Spark** : Spark is an open source cluster computing framework.
 - i. automates distribution of data and computations on a cluster of computers
 - ii. provides a fault-tolerant abstraction to distributed datasets
 - iii. is based on functional programming primitives
 - iv. provides two abstractions to data, list-like (RDDs) and table-like (Datasets)
- **2) Resilient Distributed Datasets (RDDs)** : RDDs are the core abstraction that Spark uses. RDDs make datasets distributed over a cluster of machines look like a Scala collection. RDDs:
 - i. are immutable
 - ii. reside (mostly) in memory
 - iii. are transparently distributed
 - iv. feature all FP programming primitives
 - v. in addition, more to minimize shuffling
 - vi. In practice, RDD[A] works like Scala's List[A]

Transformation: Transformation refers to the operation applied on a RDD to create new RDD. Filter, groupBy and map are the examples of transformations.

Actions: Actions refer to an operation which also applies on RDD, that instructs Spark to perform computation and send the result back to driver.

Amazon Fine Food Review Analysis : This dataset consists of reviews of fine foods from amazon. The data span a period of more than 10 years, including all ~500,000 reviews up to October 2012. Reviews include product and user information, ratings, and a plaintext review. We also have reviews from all other Amazon categories.

Source: <https://www.kaggle.com/snap/amazon-fine-food-reviews>

Introduction Amazon Food Dataset : The Amazon Fine Food Reviews dataset consists of reviews of fine foods from Amazon.

- Number of reviews: 568,454
- Number of users: 256,059
- Number of products: 74,258
- Timespan: Oct 1999 — Oct 2012
- Number of Attributes/Columns in data: 10

Attribute Information:

- Id
- ProductId — unique identifier for the product
- UserId — unique identifier for the user
- Helpfulness Numerator — number of users who found the review helpful
- HelpfulnessDenominator — number of users who indicated whether they found the review helpful or not
- Score — a rating between 1 and 5
- Time — timestamp for the review
- Summary — Brief summary of the review
- Text — Text of the review

Spark Application Code For Operations :

```
# Load Amazon Food Dataset
file1 = sc.textFile("amazon_food.csv")

# Create RDD pair
val user_id = file1.keyBy(_.UserId)
val product_id = file1.keyBy(_.ProductId)
rdd = user_id.join(product_id) // RDD Pair

# Counts the frequencies of all the pairs of products reviewed together;
frequency = rdd.count()

# Pairs of products that appear more than once and their frequencies.
The pairs of products must be sorted by frequency.
print(sorted(rdd))
```

Referances :

- 1) <https://www.analyticsvidhya.com/blog/2016/10/using-pyspark-to-perform-transformations-and-actions-on-rdd/>
- 2) <https://www.gousios.gr/courses/bigdata/spark.html>

Conclusion : In this Practical I learn to Perform Data analytics using Apache Spark on Amazon food dataset, find all the pairs of items frequently reviewed together.