Assignment No. B9

Title : Build a required Database to develop BAI tool for considering one aspect of growth to the business such as an organization of products based on expiry dates using Hadoop Ecosystem for unstructured data analytics.

Aim: A Mall has number of items for sale. Build a required Database to develop BAI tool for considering one aspect of growth to the business such as an organization of products based on expiry dates using Hadoop Ecosystem for unstructured data analytics.

Objectives:

- To study the Business intelligence.
- To explore Hadoop and Mapreduce concepts.

Theory :

Hadoop:

Hadoop is an open-source framework that allows to store and process big data in a distributed environment across clusters of computers using simple programming models. It is designed to scale up from single servers to thousands of machines, each offering local computation and storage.
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. A Hadoop frame-worked 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

Hadoop framework includes following four modules:

1. Hadoop Common: These are Java libraries and utilities required by other Hadoop modules. These libraries provides file system and OS level abstractions and contains the necessary Java _les and scripts required to start Hadoop.
2. Hadoop YARN: This is a framework for job scheduling and cluster resource management.
3. Hadoop Distributed File System (HDFS): A distributed _le system that provides high-throughput access to application data.
4. Hadoop MapReduce: This is YARN-based system for parallel processing of large data sets.

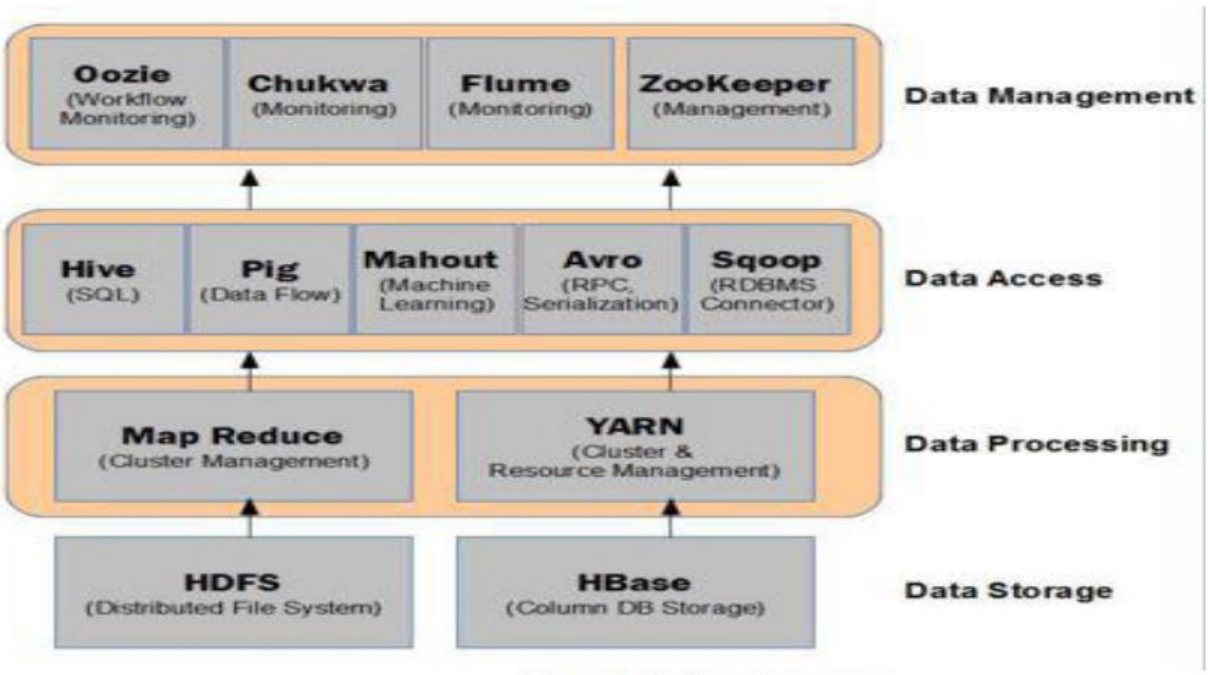

Figure 1: Hadoop Ecosystem

HDFS

Hadoop File System was developed using distributed file system design. It is run on commodity hardware. Unlike other distributed systems, HDFS is highly fault tolerant and designed using low-cost hardware.

Features of HDFS:-

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

HDFS Architecture

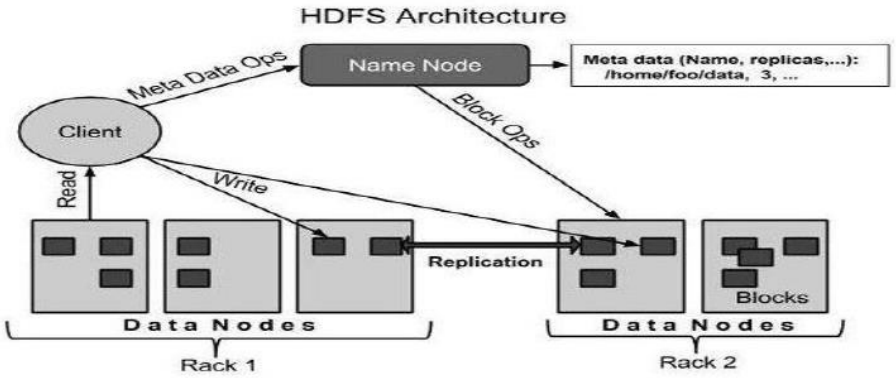

Figure 2: Hadoop Architecture

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 software that can be run on commodity hardware.
The system having the namenode acts as the master server and it does the following tasks:

1. Manages the file system namespace.
2. Regulates client's access to files.
3. 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.

Datanodes perform read-write operations on the file systems, as per client request. 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. 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. 2.Huge datasets : HDFS should have hundreds of nodes per cluster to manage the applications having huge datasets.
3. 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.

MapReduce :

MapReduce is a framework using which we can write applications to process huge amounts of data, in parallel, on large clusters of commodity hardware in a reliable manner. MapReduce is a processing technique and a program model for distributed computing based on java. The MapReduce algorithm contains two important tasks, namely Map and Reduce. Map takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (key/value pairs). Secondly, reduce task, which takes the output from a map as an input and combines those data tuples into a smaller set of tuples. As the sequence of the name MapReduce implies, the reduce task is always performed after the map job.

Mathematical Model

Let S be the solution
S={s, e, i, o, f, DD, NDD, success, failure}
s = initial state that is constructor of the class.
e = be the end state or destructor of the class.
i = input of the system.
o = output of the system.
DD-deterministic data it helps identifying the load store functions or assignment functions.
NDD- Non deterministic data of the system S to be solved. Success-desired outcome generated.
Failure-Desired outcome not generated or forced exit due to system error.
Success= {Desired outcome generated}
Failure= {Desired outcome not generated or forced exit due to system error}

Conclusion:
We have successfully implemented Hadoop and MapReduce.

Code:

PriceTypeDriver.java

```
import java.io.IOException;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
import com.pricetype.mapper.PriceTypeMapper;
import com.pricetype.reducer.PriceTypeReducer;

public class PriceTypeDriver
{
	public static void main (String[] args) throws IOException,
	ClassNotFoundException, InterruptedException
	{
		Configuration conf = new Configuration();
		Job j=new Job(conf);
		j.setJarByClass(PriceTypeDriver.class);
		j.setMapperClass(PriceTypeMapper.class);
		j.setReducerClass(PriceTypeReducer.class);
		j.setMapOutputKeyClass(Text.class);

		j.setMapOutputValueClass(LongWritable.class);
		j.setOutputKeyClass(Text.class);
		j.setOutputValueClass(LongWritable.class);
		j.setInputFormatClass(TextInputFormat.class);
		j.setOutputFormatClass(TextOutputFormat.class);
		Path input = new Path(args[0]);
		Path output = new Path(args[1]);
		FileInputFormat.addInputPath(j, input);
		FileOutputFormat.setOutputPath(j, output);
		boolean isJobRunning = j.waitForCompletion(true);
		System.exit(isJobRunning ? 0 : 1);
	}
}
```

PriceTypeMapper.java

```
import java.io.IOException;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
public class PriceTypeMapper extends Mapper<LongWritable,Text,Text,LongWritable>
{
	@Override
	protected void map(LongWritable key, Text value, Context context)throws
	IOException, InterruptedException
	{
		LongWritable one = new LongWritable(1);
		String line = value.toString();
		String [] words = line.split("\\|");
		if(words.length>=3)
		{
			if(words[1].equals("R"))
			{
```

```
                                context.write(value,one);
                        }
                }
        }
}
```

PriceTypeReducer.java

```
import java.io.IOException;
import java.util.Date;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;
public class PriceTypeReducer extends Reducer<Text,LongWritable,Text,Text>
{
        @SuppressWarnings("deprecation")
        @Override
        protected void reduce(Text key, Iterable<LongWritable> values,Context context)
        throws IOException, InterruptedException
        {
                String line = key.toString();
                String [] words = line.split("\\|");
                Date startDate = new Date(words[3]);
                long differ = (System.currentTimeMillis() - startDate.getTime());
                long diff_h=differ/(60*60*1000);
                long days=diff_h/24;
                if(days > 365)
                {
                        context.write(new Text(line), new Text(" "));
                }
        }
}
```

Commands

To create input file:
1. vim in.txt
2. To add records: type i (i-insert mode)
3. To save and exit from vim, press escape key- : - wq and hit enter
4. hadoop fs - copyFromLocal MyProject.jar
5. hadoop jar MyProject.jar myproject.com.driver.PriceTypeDriver in.txt output125
6. Output is generated in directory output125
7. hadoop fs -ls output125
8. hadoop fs -cat output125/part-r-00000

Input

```
1|R|10|01/01/2015

2|R|20|01/02/2016

3|R|30|01/02/2000

4|R|40|01/03/2004
```

Output

```
[cloudera@quickstart ~]$ vim in.txt

[cloudera@quickstart ~]$ hadoop jar MyProject.jar
myproject.com.driver.PriceTypeDriver
in.txt output125
16/03/16 01:04:03
```

```
INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
16/03/16 01:04:04
WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not
performed.
Implement the Tool interface and execute your application with ToolRunner to
remedy this.
16/03/16 01:04:05
INFO input.FileInputFormat: Total input paths to process : 1
16/03/16 01:04:05
INFO mapreduce.JobSubmitter: number of splits:1
16/03/16 01:04:05
INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1457534540297_0028
16/03/16 01:04:06

INFO impl.YarnClientImpl: Submitted application application_1457534540297_0028
16/03/16 01:04:06
INFO mapreduce.Job: The url to track the job:
http://quickstart.cloudera:8088/proxy/
application_1457534540297_0028
16/03/16 01:04:06
INFO mapreduce.Job: Running job: job_1457534540297_0028
16/03/16 01:04:17
INFO mapreduce.Job: Job job_1457534540297_0028 running in uber mode : false
16/03/16 01:04:17
INFO mapreduce.Job: map 0% reduce 0%
16/03/16 01:04:35
INFO mapreduce.Job: map 100% reduce 0%
16/03/16 01:04:55
INFO mapreduce.Job: map 100% reduce 100%
16/03/16 01:04:56
INFO mapreduce.Job: Job job_1457534540297_0028 completed successfully
16/03/16 01:04:56
INFO mapreduce.Job: Counters: 49
File System Counters
FILE: Number of bytes read=118
FILE: Number of bytes written=224531
FILE: Number of read operations=0
FILE: Number of large read operations=0
FILE: Number of write operations=0
HDFS: Number of bytes read=189
HDFS: Number of bytes written=60
HDFS: Number of read operations=6
HDFS: Number of large read operations=0
HDFS: Number of write operations=2
Job Counters
Launched map tasks=1
Launched reduce tasks=1

Data-local map tasks=1
Total time spent by all maps in occupied slots (ms)=16314
Total time spent by all reduces in occupied slots (ms)=16099
Total time spent by all map tasks (ms)=16314
Total time spent by all reduce tasks (ms)=16099
Total vcore-seconds taken by all map tasks=16314
Total vcore-seconds taken by all reduce tasks=16099
Total megabyte-seconds taken by all map tasks=16705536
Total megabyte-seconds taken by all reduce tasks=16485376
Map-Reduce Framework
Map input records=4
```

```
Map output records=4
Map output bytes=104
Map output materialized bytes=118
Input split bytes=117
Combine input records=0
Combine output records=0
Reduce input groups=4
Reduce shuffle bytes=118
Reduce input records=4
Reduce output records=3
Spilled Records=8
Shuffled Maps =1
Failed Shuffles=0
Merged Map outputs=1
GC time elapsed (ms)=185
CPU time spent (ms)=3600
Physical memory (bytes) snapshot=335437824
Virtual memory (bytes) snapshot=3008466944
Total committed heap usage (bytes)=226562048
Shuffle Errors
BAD_ID=0
CONNECTION=0
IO_ERROR=0
WRONG_LENGTH=0
WRONG_MAP=0
WRONG_REDUCE=0
File Input Format Counters
Bytes Read=72
File Output Format Counters
Bytes Written=60
[cloudera@quickstart ~]$ hadoop fs -ls output125
Found 2 items
-rw-r--r-- 1
cloudera cloudera 0 2016-03-16 01:04 output125/_SUCCESS
-rw-r--r-- 1 cloudera cloudera 60 2016-03-16 01:04

output125/part-r-00000
[cloudera@quickstart ~]$ hadoop fs -cat output125/part-r-00000
1|R|10|01/01/2015
3|R|30|01/02/2000
4|R|40|01/03/2004
```