Department of Computer Science and Engineering

Cloud Architecture

Lab10A: Unified Platform for Big Data Processing

UCID:
Student Name:
Branch:

Objective: Setting up and utilizing Apache Hadoop and Apache Spark to process large datasets efficiently.

This lab focuses on understanding the integration of Hadoop's distributed storage with Spark's fast computation framework, enabling a unified platform for big data processing.

Outcomes: After successful completion of the lab, students should be able to:

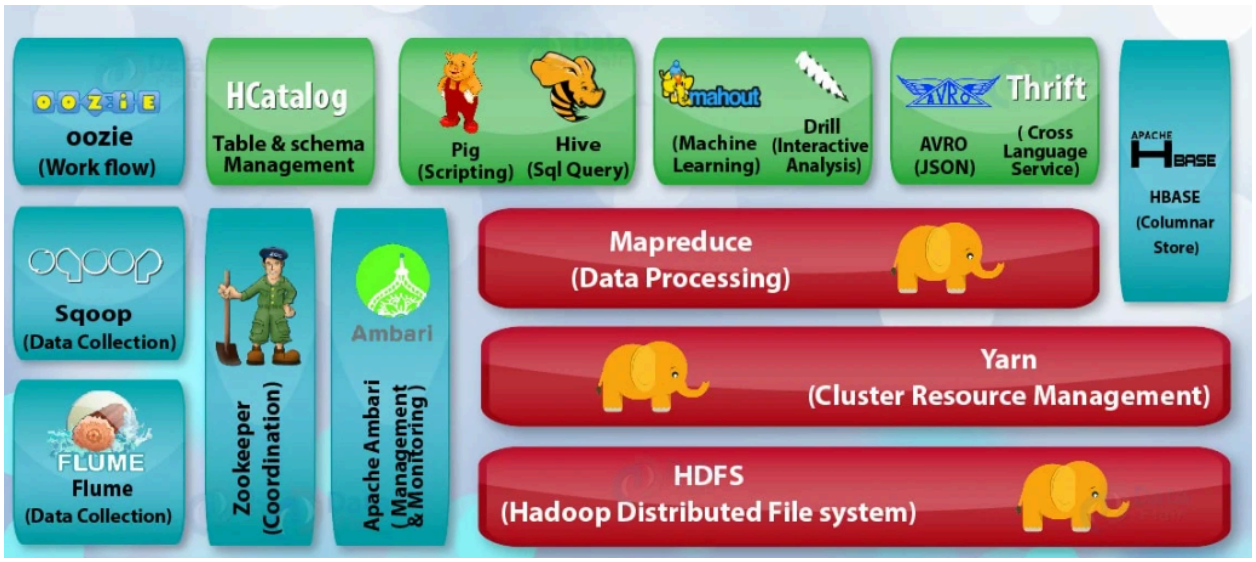
Upon completion of this lab, students will be able to:
[1] Install and configure Apache Hadoop and Apache Spark on a distributed cluster.
[2] Perform data storage and retrieval operations using the Hadoop Distributed File System (HDFS).
[3] Use Apache Spark to perform data analysis and processing on datasets stored in HDFS.
[4] Understand the integration of Hadoop with Spark and run Spark jobs on the Hadoop platform for unified big data processing.

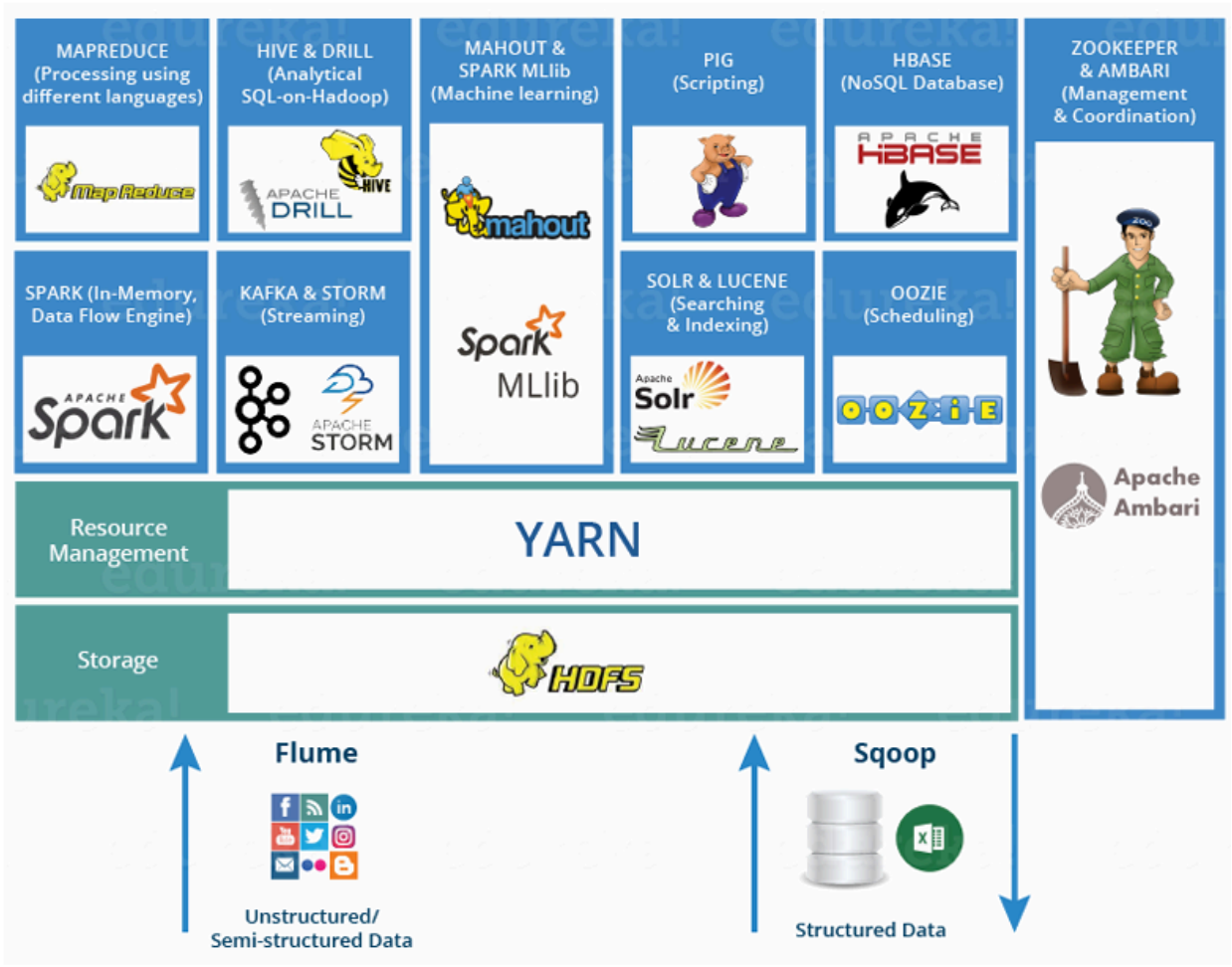
System Requirements:

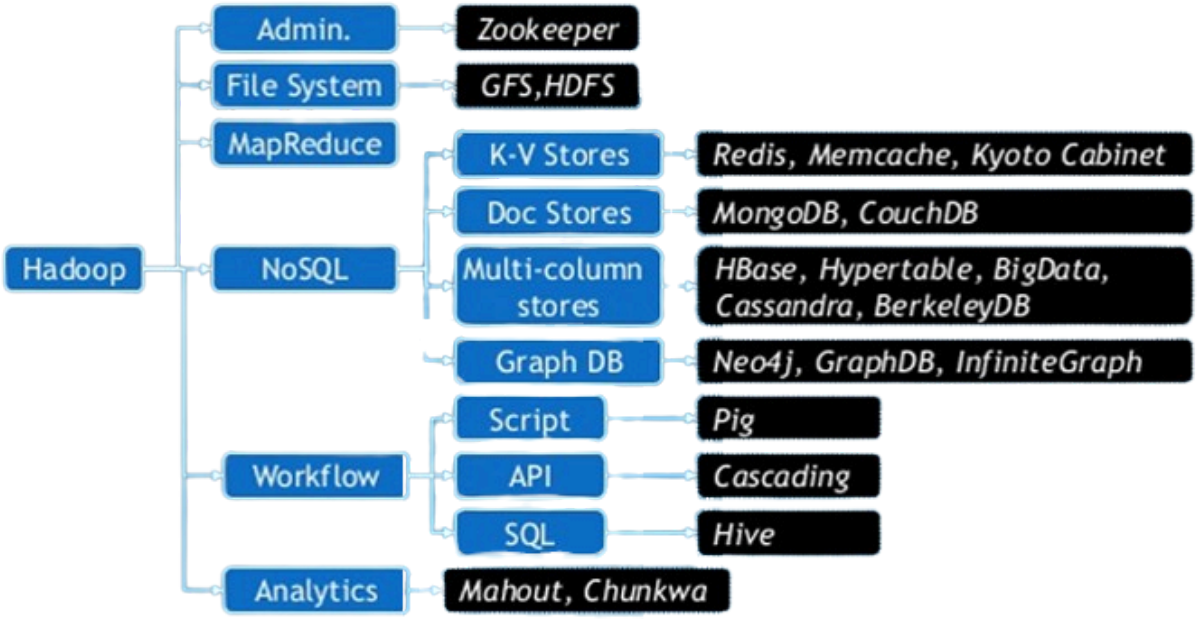
The following minimum prerequisites are required for OpenStack:
Operating System: Ubuntu 20.04 LTS or later, or CentOS 7/8
Java Development Kit (JDK): JDK 8 or higher
Apache Hadoop: Version 3.3.0 or later
Apache Spark: Version 3.0.0 or later
RAM: Minimum 4 GB per node (8 GB recommended)
Disk Space: Minimum 10 GB per node
Python: Version 3.6 or higher for Spark Python API (PySpark)
Network Configuration: SSH and Internet connectivity for downloading packages

Brief Introduction to Big Data Analytics : (Write in your own words)

Draw a neat Architecture diagram:




MAPREDUCE (Processing using different languages)
HIVE & DRILL (Analytical SQL-on-Hadoop)
MAHOUT & SPARK MLlib (Machine learning)
PIG (Scripting)
HBASE (NoSQL Database)
ZOOKEEPER & AMBARI (Management & Coordination)
SPARK (In-Memory, Data Flow Engine)
KAFKA & STORM (Streaming)
SOLR & LUCENE (Searching & Indexing)
OOZIE (Scheduling)
Resource Management
YARN
Storage
HDFS
Flume
Unstructured/ Semi-structured Data
Sqoop
Structured Data


Hadoop
Admin.
Zookeeper
File System
GFS,HDFS
MapReduce
NoSQL
K-V Stores
Redis, Memcache, Kyoto Cabinet
Doc Stores
MongoDB, CouchDB
Multi-column stores
HBase, Hypertable, BigData, Cassandra, BerkeleyDB
Graph DB
Neo4j, GraphDB, InfiniteGraph
Workflow
Script
Pig
API
Cascading
SQL
Hive
Analytics
Mahout, Chunkwa

Procedure:

Refer [1] [2] [3]

Step-1: Setup Apache Hadoop using CDH
Apache Hadoop (CDH 5.8) Install with QuickStarts Docker

```
cse-406a@CSE-406A:~$ docker ps
CONTAINER ID   IMAGE                COMMAND                  CREATED             STATUS             PORTS
                                                                                 NAMES
532c7229bb99   cloudera/quickstart   "/usr/bin/docker-qui…"   About an hour ago   Up About an hour   0.0.0.0:80->80/tcp, :::80->80/tcp, 0.0.0.0
:7180->7180/tcp, :::7180->7180/tcp, 0.0.0.0:8888->8888/tcp, :::8888->8888/tcp   sad_cerf
cse-406a@CSE-406A:~$
```

Step-2: Hadoop Running a MapReduce Job (Single-Node Cluster)

```
cse-406a@CSE-406A:~$ docker search cloudera/quickstart
NAME                                      DESCRIPTION                                     STARS     OFFICIAL   AUTOMATED
cloudera/quickstart                       Single-node deployment of Cloudera's 100% op…   413
cloudera/clusterdock                      Single-host, multi-node deployment of CDH 5.…   120
seabreeze/azure-mesh-quickstart-web       Azure Service Fabric Mesh quickstart web fro…   1                     [OK]
seabreeze/azure-mesh-quickstart-data      Azure Service Fabric Mesh quickstart data ba…   1                     [OK]
```

1) WordCounter MapReducer:

```
[root@quickstart /]# hdfs dfs -ls /user/cloudera/hadoop
Found 2 items
-rw-r--r--   1 root cloudera         83 2024-11-07 10:27 /user/cloudera/hadoop/input.txt
drwxr-xr-x   - root cloudera          0 2024-11-07 10:40 /user/cloudera/hadoop/output.txt
[root@quickstart /]# hdfs dfs -ls /user/cloudera/hadoop/output.txt
Found 2 items
-rw-r--r--   1 root cloudera          0 2024-11-07 10:40 /user/cloudera/hadoop/output.txt/_SUCCESS
-rw-r--r--   1 root cloudera         95 2024-11-07 10:40 /user/cloudera/hadoop/output.txt/part-r-00000
[root@quickstart /]# hdfs dfs -cat /user/cloudera/hadoop/output.txt/part-r-00000
        1
Hadoop  3
Hello   1
Welcome 1
data    1
files.  1
for     1
is      1
large   1
of      1
processing      1
to      1
used    1
[root@quickstart /]# cat input.txt
Welcome to Hadoop
Hello Hadoop
Hadoop is used for processing of large data files.
```

2) Sales calculation by country MapReducer Program:

Mapper Class :

```
import
org.apache.hadoop.io.DoubleWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;

import

org.apache.hadoop.mapreduce.Mapper;

import java.io.IOException;


public class SalesMapper extends Mapper<LongWritable, Text, Text, DoubleWritable> {



        private static final int COUNTRY_INDEX = 7; // Assuming country is the 8th column (index
        7) private static final int PRICE_INDEX = 2; // Assuming price is the 3rd column (index 2)


        @Override

protected void map(LongWritable key, Text value, Context context) throws IOException,
InterruptedException {

        String line = value.toString();



        // Split CSV by comma
        String[] fields =
        line.split(",");
```

```
        if (fields.length > COUNTRY_INDEX && fields.length >
        PRICE_INDEX) { try {

        String country = fields[COUNTRY_INDEX];

        double price = Double.parseDouble(fields[PRICE_INDEX]);


        context.write(new Text(country), new DoubleWritable(price));
        } catch (NumberFormatException e) {
        // Ignore rows with invalid data
        }
        }
        }
}
```

Reducer Class:

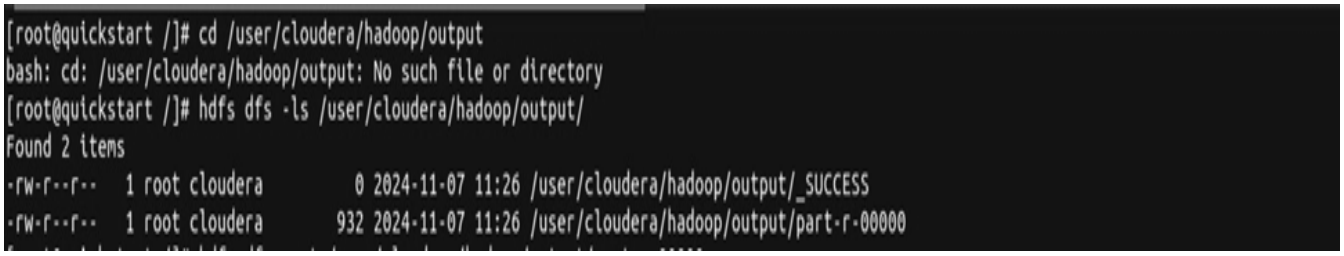
```
import
org.apache.hadoop.io.DoubleWritable;
import org.apache.hadoop.io.Text;
import

org.apache.hadoop.mapreduce.Reducer;

import java.io.IOException;


public class SalesReducer extends Reducer<Text, DoubleWritable, Text, DoubleWritable> {



        @Override
protected void reduce(Text key, Iterable<DoubleWritable> values, Context context) throws
IOException, InterruptedException {

        double totalSales = 0;
```

```
        // Sum all sales values for each
        country for (DoubleWritable value :
        values) { totalSales += value.get();

        }


        context.write(key, new DoubleWritable(totalSales));
        }
}
```

2) Driver class:

```
import
org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import
org.apache.hadoop.io.DoubleWritable;
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.output.FileOutputFormat;


public class SalesAnalysis {


        public static void main(String[] args) throws
        Exception { if (args.length != 2) {
        System.err.println("Usage: SalesAnalysis <input path> <output
        path>"); System.exit(-1);

        }



        Configuration conf = new Configuration();

        Job job = Job.getInstance(conf, "Sales Analysis");
```

```
        job.setJarByClass(SalesAnalysis.class);
        job.setMapperClass(SalesMapper.class);
        job.setReducerClass(SalesReducer.class);


        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(DoubleWritable.class);


        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));


        System.exit(job.waitForCompletion(true) ? 0 : 1);
        }
}
```



```
[root@quickstart /]# hdfs dfs -cat /user/cloudera/hadoop/output/part-r-00000
Argentina       1200.0
Australia       64800.0
Austria 10800.0
Bahrain 1200.0
Belgium 12000.0
Bermuda 1200.0
Brazil  12300.0
Bulgaria        1200.0
Canada  124800.0
Cayman Isls     1200.0
China   1200.0
Costa Rica      1200.0
Czech Republic  6000.0
Denmark 18000.0
Dominican Republic      1200.0
Finland 2400.0
France  53100.0
Germany 42000.0
Greece  1200.0
Guatemala       1200.0
Hong Kong       1200.0
Hungary 3600.0
Iceland 1200.0
India   2400.0
Ireland 69900.0
Israel  1200.0
Italy   37800.0
Japan   2400.0
Jersey  1200.0
Kuwait  1200.0
Latvia  1200.0
Luxembourg      1200.0
Malaysia        1200.0
```

```
India   2400.0
Ireland 69900.0
Israel  1200.0
Italy   37800.0
Japan   2400.0
Jersey  1200.0
Kuwait  1200.0
Latvia  1200.0
Luxembourg      1200.0
Malaysia        1200.0
Malta   4800.0
Mauritius       3600.0
Moldova 1200.0
Monaco  2400.0
Netherlands     44700.0
New Zealand     7200.0
Norway  21600.0
Philippines     2400.0
Poland  2400.0
Romania 1200.0
Russia  3600.0
South Africa    12300.0
South Korea     1200.0
Spain   16800.0
Sweden  22800.0
Switzerland     76800.0
Thailand        4800.0
The Bahamas     2400.0
Turkey  7200.0
Ukraine 1200.0
United Arab Emirates    12000.0
United Kingdom  144000.0
United States   737000.0
[root@quickstart /]#
```

Conclusion:

In conclusion, the experiment with Hadoop demonstrated how deploying it in a Docker environment can simplify installation and dependency management while maintaining flexibility and efficiency. By running two MapReduce tasks—one for word counting and another for analyzing a sales dataset—we explored Hadoop's ability to handle diverse data processing needs.

The word count task highlighted the core principles of MapReduce, while the sales analysis showed its practical use in processing real-world structured data. Overall, this experiment reinforced Hadoop's strengths in distributed data processing and its adaptability to a wide range of workloads.

References:

[1] https://www.bogotobogo.com/Hadoop/BigData_hadoop_CDH5.8_QuickStarts_Docker_Install.php

[2]Spark and Hadoop Installation Guide (Medium article):
https://medium.com/apache-spark/spark-on-hadoop-getting-started-guide-9a2f3b102c56

[3] Big Data Processing using Hadoop and Spark (Comprehensive guide):
https://towardsdatascience.com/big-data-processing-with-hadoop-and-spark-5e2bda9e0d38