SCHOOL OF COMPUTER SCIENCE AND ENGINEERING (SCOPE)

PARALLEL AND DISTRIBUTED COMPUTING (CSE 4001)
J-COMPONENT REPORT

IMPLEMENTING WORD COUNT USING MAP REDUCED PROGRAM IN HADOOP – SINGLE NODE CLUSTER IN UBUNTU

TEAM MEMBERS:

ADITYA ROHILLA (18BCE0929)
KHWAAB THAREJA (18BCE0930)
PRATHIK PUGAZHENTHI (18BCE0940)
DVR ADITYA (18BCE0949)
STUTI TIWARI (18BCE0950)
RAHUL ANAND (18BCE0953)

SUBMITTED TO:
PROF. GOPICHAND G.
(SLOT : C2)

- **INDEX:**

- **ABSTRACT:**

MapReduce is a programming model and an associated implementation for processing and generating large data sets. Users specify a map function that processes a key/value pair to generate a set of intermediate key/value pairs, and a reduce function that merges all intermediate values associated with the same intermediate key. Many real world tasks are expressible in this model. Programs written in this functional style are automatically parallelized and executed on a large cluster of commodity machines. The run-time system takes care of the details of partitioning the input data, scheduling the program's execution across a set of machines, handling machine failures, and managing the required inter-machine communication. This allows programmers without any experience with parallel and distributed systems to easily utilize the resources of a large distributed system.

- **INTRODUCTION:**

Hadoop MapReduce is a software framework for easily writing applications which process vast amounts of data (multi-terabyte data-sets) in-parallel on large clusters (thousands of nodes) of commodity hardware in a reliable, fault-tolerant manner. A MapReduce *job* usually splits the input data-set into independent chunks which are processed by the *map tasks* in a completely parallel manner. The framework sorts the outputs of the maps, which are then input to the *reduce tasks*. Typically, both the input and the output of the job are stored in a file-system. The framework takes care of scheduling tasks, monitoring them and re-executes the failed tasks. The MapReduce framework consists of a single master ResourceManager, one worker NodeManager per cluster-node, and MRAppMaster per application.

Minimally, applications specify the input/output locations and supply *map* and *reduce* functions via implementations of appropriate interfaces and/or abstract-classes. These, and other job parameters, comprise the *job configuration*. The Hadoop *job client* then submits the job (jar/executable etc.) and configuration to the ResourceManager which then assumes the responsibility of distributing the software/configuration to the workers, scheduling tasks and monitoring them, providing status and diagnostic information to the job-client.

- **MODULES USED:**

  - **Module 1** : creating namenode , Data inputs and Mapper class
  - **Module 2** : Reducer class , Driver Class and Graphical Representation of Data

- **HADOOP INSTALLATION STEPS:**

Steps to Install Apache Hadoop 2.7.0 Single Node Cluster on Ubuntu 16.04 are as follows:

- **Part- A Setting up Ubuntu Server Machine for Hadoop**

Step 1: Login with Root
#su -
password:

Step 2: Update the System
#apt-get update

Step 3:

Installation of OpenSSH Server
#apt-get install openssh-server

Step 4:

After the SSH server is installed, configuration can be done by editing sshd_config that resides in the /etc/ssh directory. It is important to note sshd_config is for the SSH server while ssh_config is for the SSH client. Create a backup copy of sshd_config that can be used to restore your configuration by running the command below.
#cp /etc/ssh/sshd_config /etc/ssh/sshd_config.factory-defaults

Step 5:

Open sshd_config in a text editor by running the command below for Ubuntu versions

#gedit /etc/ssh/sshd_config

Disable password authentication by changing this line in the configuration file #PasswordAuthentication yes to PasswordAuthentication no.

Add the following Lines to the End:

AllowUsers stuti
PermitRootLogin no
PubkeyAuthentication yes

Change LogLevel INFO to LogLevel VERBOSE.

Save and Restart SSH

#systemctl restart ssh

Step 6:

After SSH is configured, an SSH key for the user eduonix is generated by running the commands below. They create an RSA key pair without a password. A password would be required every time Hadoop interacts with its nodes so we can save ourselves the bother of being prompted for a password every time.

#su - stuti
#ssh-keygen -t rsa -P ""

After the key has been created, we use it to enable SSH access to the local machine by running the command below which adds it to the list of known keys.

#cat $HOME/.ssh/id_rsa.pub >> $HOME/.ssh/authorized_keys Testing

SSH

#ssh localhost

- **Part-B Installing Apache Hadoop**

Step 1:

Adding Jave Repository and Installing Oracle Java 8
#add-apt-repository ppa:webupd8team/java #apt-get
update

#apt-get install oracle-java8-installer
#java -version

Step 2:

Downloading and Installing Hadoop
Login with user "stuti"

#cd Desktop
#sudo tar xzvf hadoop-2.7.0.tar.gz
#sudo mkdir /usr/local/hadoop

#sudo mv hadoop-2.7.0 /usr/local/hadoop

#sudo chown -R stuti /usr/local/Hadoop

Step 3:

You need to edit .bashrc file for the user stuti, so open it in a text editor by running gedit ~/.bashrc from a terminal.

Check Where the Java is Installed
#readlink -f /usr/bin/java

#gedit ~/.bashrc Type

the following:

export JAVA_HOME=/usr/lib/jvm/java-8-oracle

export HADOOP_INSTALL=/usr/local/hadoop/hadoop-2.7.0 export PATH=$PATH:$HADOOP_INSTALL/bin export PATH=$PATH:$HADOOP_INSTALL/sbin

export HADOOP_MAPRED_HOME=$HADOOP_INSTALL
export HADOOP_COMMON_HOME=$HADOOP_INSTALL
export HADOOP_HDFS_HOME=$HADOOP_INSTALL

export YARN_HOME=$HADOOP_INSTALL export

HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_INSTALL/lib/nati

ve
export HADOOP_OPTS="- Djava.library.path=$HADOOP_INSTALL/lib/native"

Save the file and exit

Save the Changes to Bashrc File #source

~/.bashrc

Step 4:

Exporting JAVA_HOME Path

The directory /usr/local/hadoop/hadoop-2.7.0/etc/hadoop contains configuration files. Open hadoop-env.sh in a text editor and set JAVA_HOME variable by adding the line below. This specifies the java installation that will be used by Hadoop.

#cd /usr/local/hadoop/hadoop-2.7.0/etc/hadoop #nano
hadoop-env.sh

export JAVA_HOME=/usr/lib/jvm/java-8-oracle save
and exit

Step 5:

Create a directory that will act as the base for other temporary directories and assign it to the user stuti by running the commands below.

#sudo mkdir -p /app/hadoop/tmp #sudo

chown stuti /app/hadoop/tmp

- **Part-C Editing the XML Based Hadoop Configuration Files**

All the Hadoop configuration files reside under
usr/local/hadoop/hadoop-2.7.0/etc/hadoop

#cd /usr/local/hadoop/hadoop-2.7.0/etc/hadoop

A. Edit Core-site.xml

<property>
<name>fs.default.name</name>
<value>hdfs://localhost:9000</value>
</property>

B. Edit mapred-site.xml

#cp /usr/local/hadoop/hadoop-2.7.0/etc/hadoop/mapred-site.xml.template /usr/local/hadoop/hadoop-2.7.0/etc/hadoop/mapred- site.xml
#nano mapred-site.xml

<property>
<name>mapred.job.tracker</name>
<value>localhost:9001</value>
</property>
<property>
<name>mapreduce.framework.name</name>
<value>yarn</value>
</property>

C. Edit yarn-site.xml

#nano yarn-site.xml
<property>
<name>yarn.nodemanager.aux-services</name>
<value>mapreduce_shuffle</value>
</property>
<property>
<name>yarn.nodemanager.aux- services.mapreduce.shuffle.class</name>

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

- **Part-D**

The hdfs-site.xml is used to specify the namenode and datanode directories. Before modifying this file, we create the namenode and datanode directories.

#sudo mkdir -p /usr/local/hadoop_store/hdfs/namenode #sudo mkdir -p /usr/local/hadoop_store/hdfs/datanode #sudo chown -R stuti /usr/local/hadoop_store

Modify hdfs-site.xml

#nano hdfs-site.xml

<property>

<name>dfs.replication</name>

<value>4</value>

</property>

<property>

<name>dfs.namenode.name.dir</name>

<value> file:/usr/local/hadoop_store/hdfs/namenode </value>

</property>

<property>

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

<value> file:/usr/local/hadoop_store/hdfs/datanode </value>

</property>

Save and Exit

- **Part-E Final Steps**

Format the file system by running hdfs namenode -format to initialize the file system

#hdfs namenode -format Start

the single node cluster #start-

dfs.sh

#start-yarn.sh

#jps

Open the Web Browser http://ipaddress:8088 (Main Cluster) http://ipaddress:50070 (Detailed Information)

- **HADOOP ARCHITECTURE:**

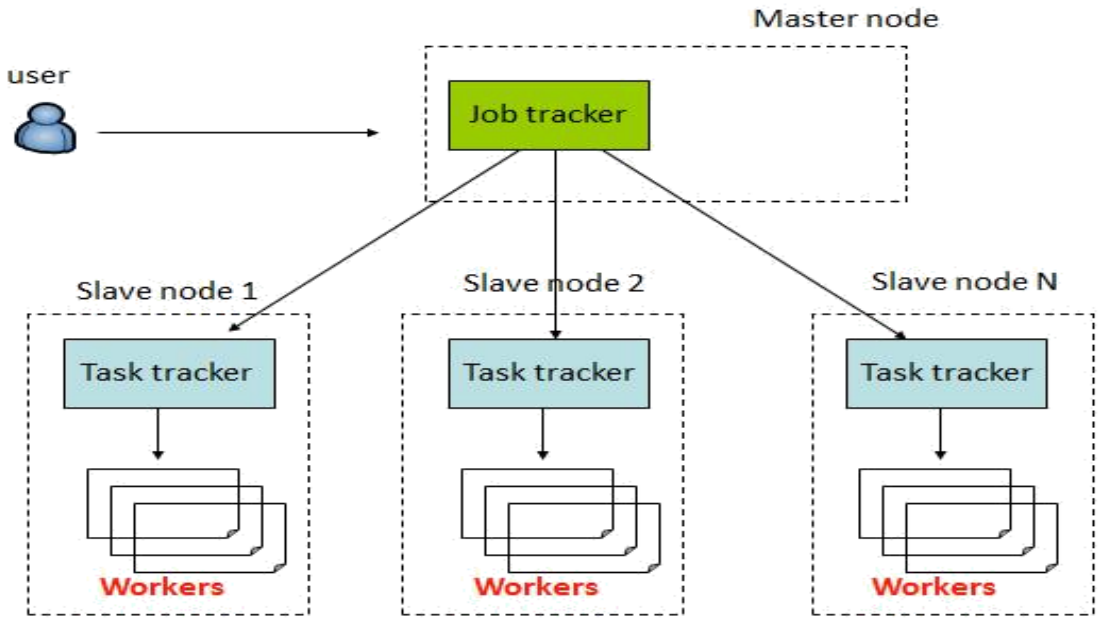
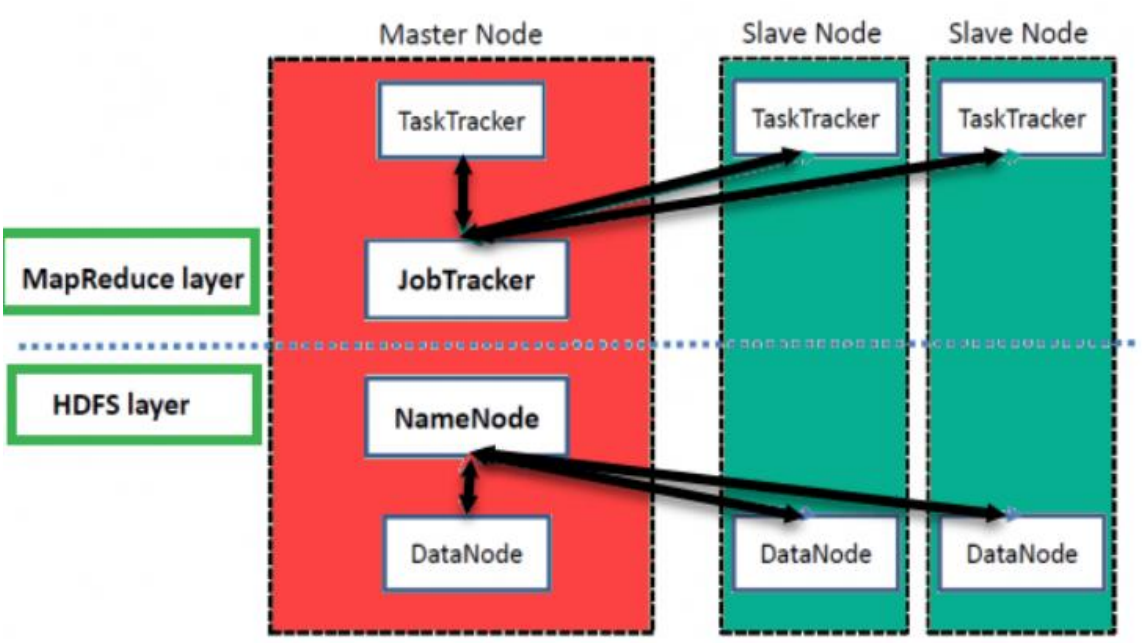


Figure 1. Overview of Architecture

- Hadoop has a Master-Slave Architecture for data storage and distributed data processing using MapReduce and HDFS methods.



High level Hadoop Architecture

- **WORKING EXPLAINED:**

1. JOB SCHEDULING :

✓ Hadoop divides the job into tasks. There are two types of tasks:

a. **Map tasks** (Spilts & Mapping)

b. **Reduce tasks** (Shuffling, Reducing)

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

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

b. **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**.

✓ A job is divided into multiple tasks which are then run onto multiple data nodes in a cluster.

✓ It is the responsibility of jobtracker to coordinate the activity by scheduling tasks to run on different data nodes.

✓ Execution of individual task is then looked after by tasktracker, which resides on every data node executing part of the job.

✓ Tasktracker's responsibility is to send the progress report to the jobtracker.

✓ In addition, tasktracker periodically sends **'heartbeat'** signal to the Jobtracker to notify him of current state of the system.

✓ Jobtracker keeps track of overall progress of each job.

- ✓ In the event of task failure, the jobtracker can reschedule it on a different tasktracker.

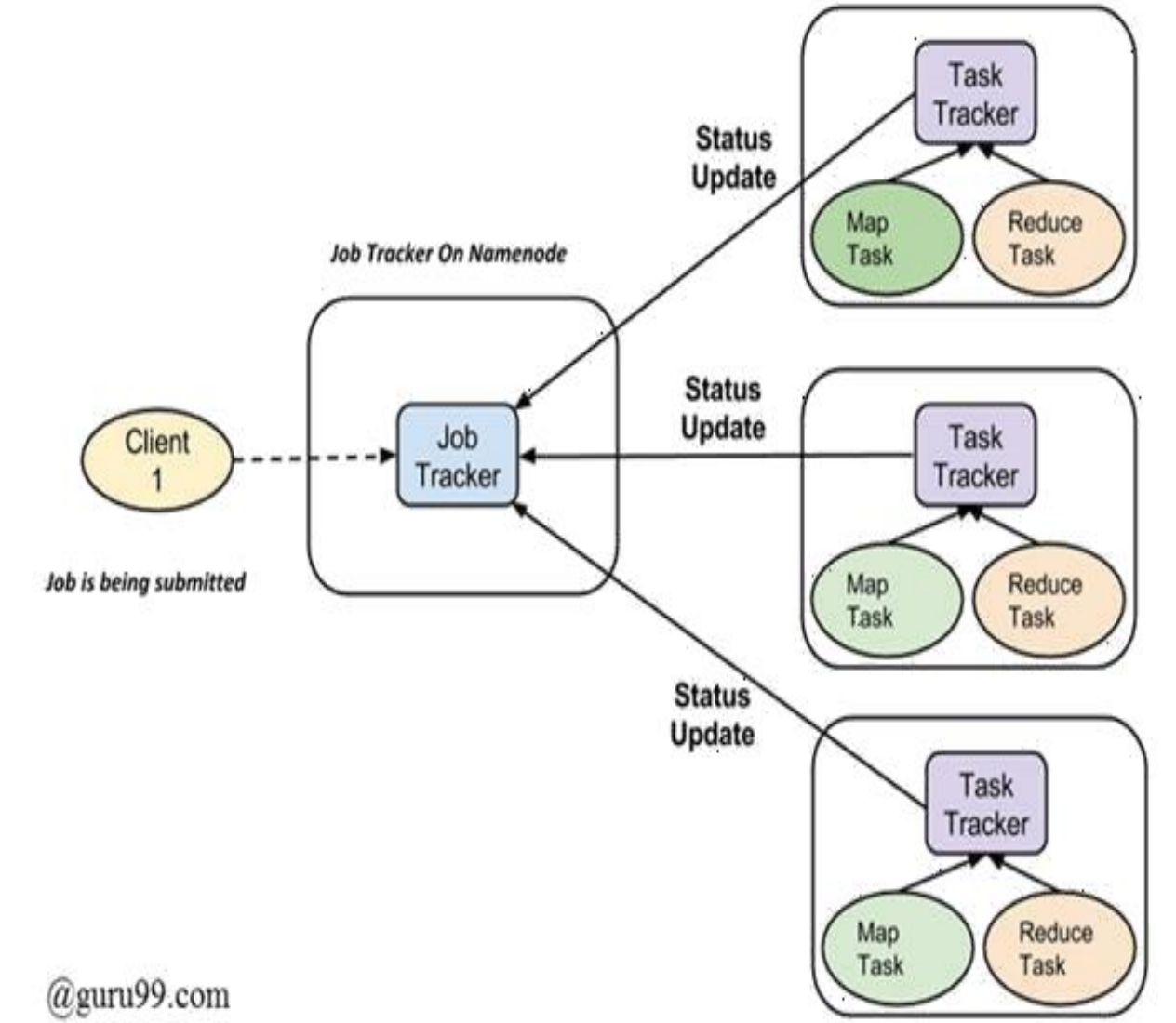
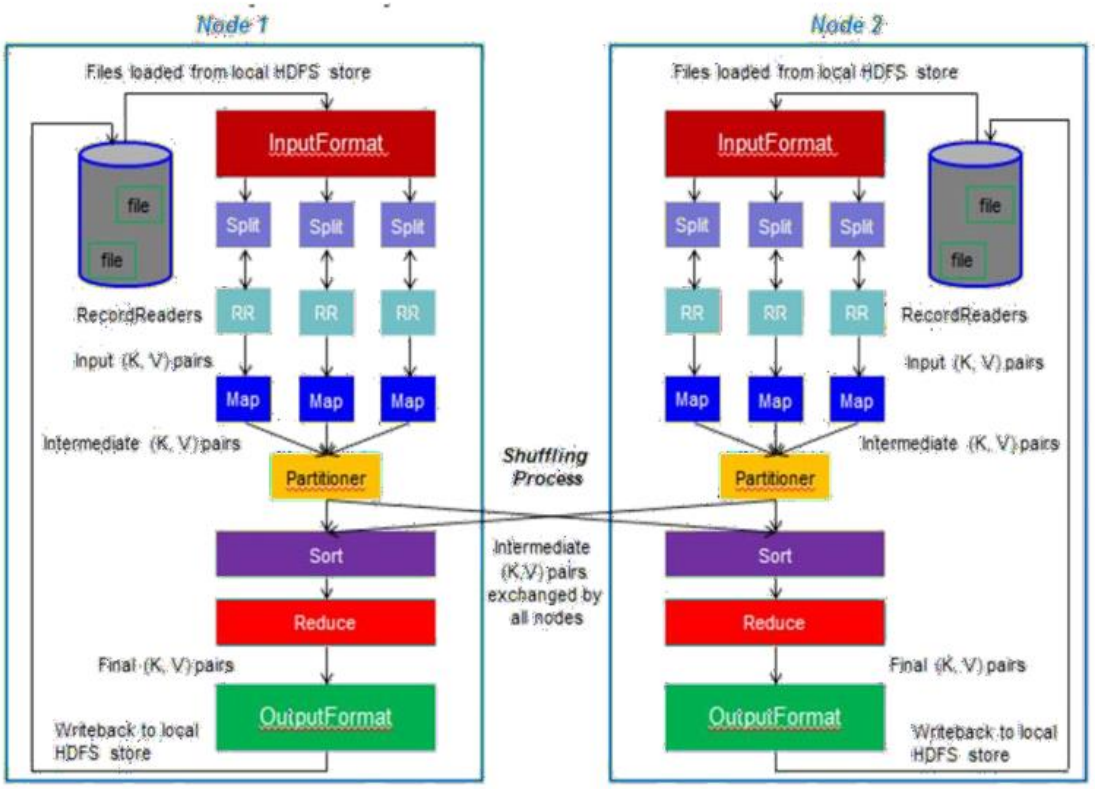


Figure 2. Job- Scheduling

- **COMPONENTS IN DETAIL:**



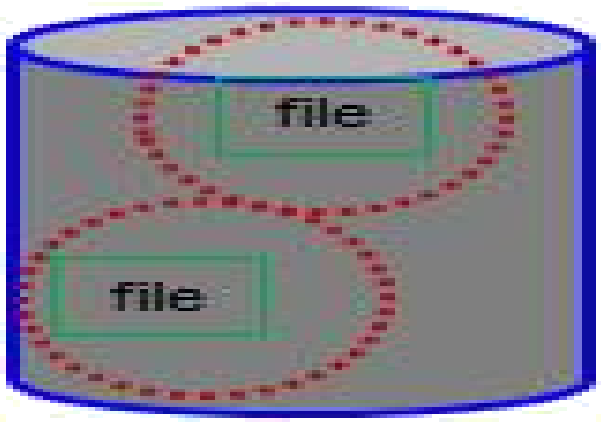
1. KEYS AND VALUES :

- The programmer in MapReduce has to specify two functions, the *map function* and *reduce function* that implement the Mapper and the Reducer in a MapReduce program.
- In MapReduce data elements are always structured as key-value (i.e., (K, V)) pairs.
- The map and reduce functions receive and *emit* (K, V) pairs.

2. INPUT FILES :

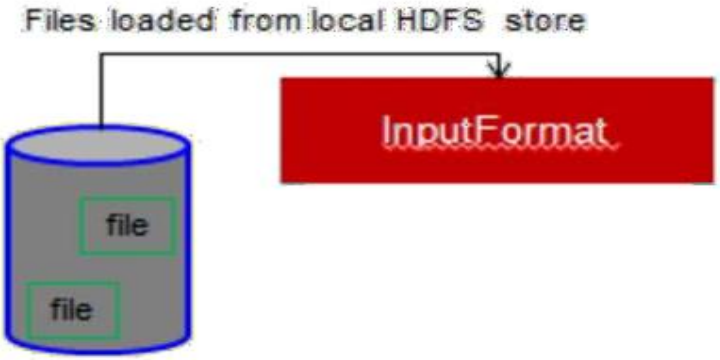
- *Input files* are where the data for a MapReduce task is initially stored.
- The input files typically reside in a distributed file system (e.g. HDFS).
- The format of input files is arbitrary.

1. Line-based log files
2. Binary files
3. Multi-line input records
4. Or something else entirely



3. INPUT FORMAT :

- How the input files are split up and read is defined by the *InputFormat.*
- InputFormat is a class that does the following:
  - Selects the files that should be used for input
  - Defines the *InputSplits* that Will break a file
  - Provides a factory for *RecordReader* objects that read the file

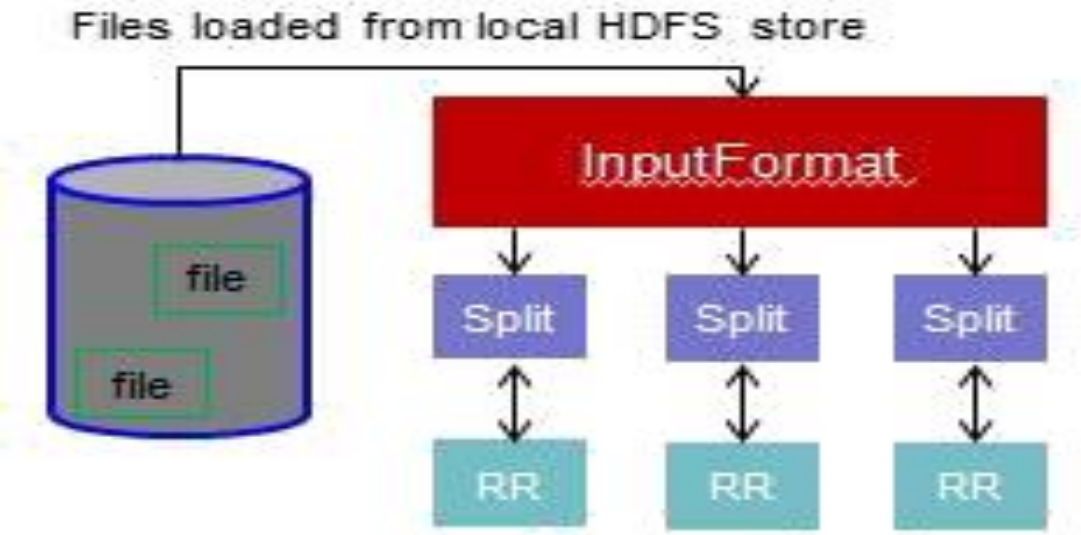


4. INPUT SPLITS :

- An *input split* describes a unit of work that comprises a single map task in a MapReduce program.
- By default, InputFormat breaks a file up into 64MB splits
- By dividing the file into splits, we allow several map tasks to operate on a single file in parallel
- If the file is very large, this can Improve performance significantly through parallelism Each map task corresponds to a *single* input split

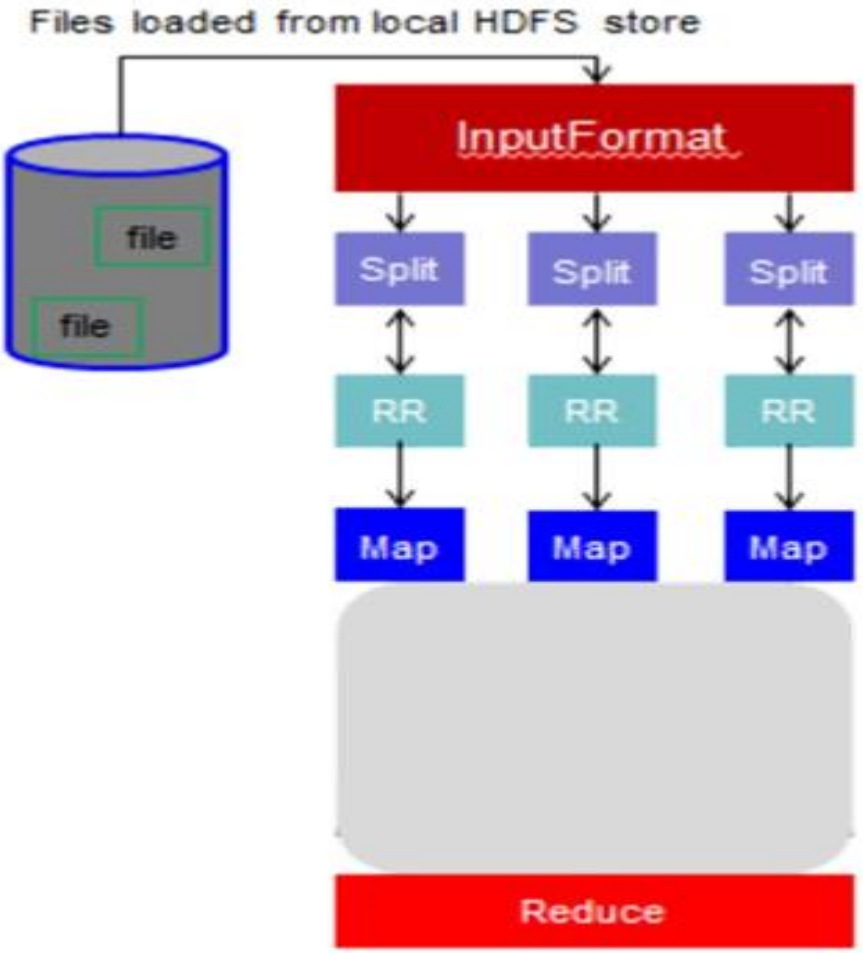
5. RECORD READER

- The input split defines a slice of work but does not describe how to access it.
- The *RecordReader* class actually loads data from its source and converts it into (K, V) pairs suitable for reading by Mappers
- The RecordReader is invoked repeatedly on the input until the entire split is consumed
- Each invocation of the RecordReader leads to another call of the map function defined by the programmer.



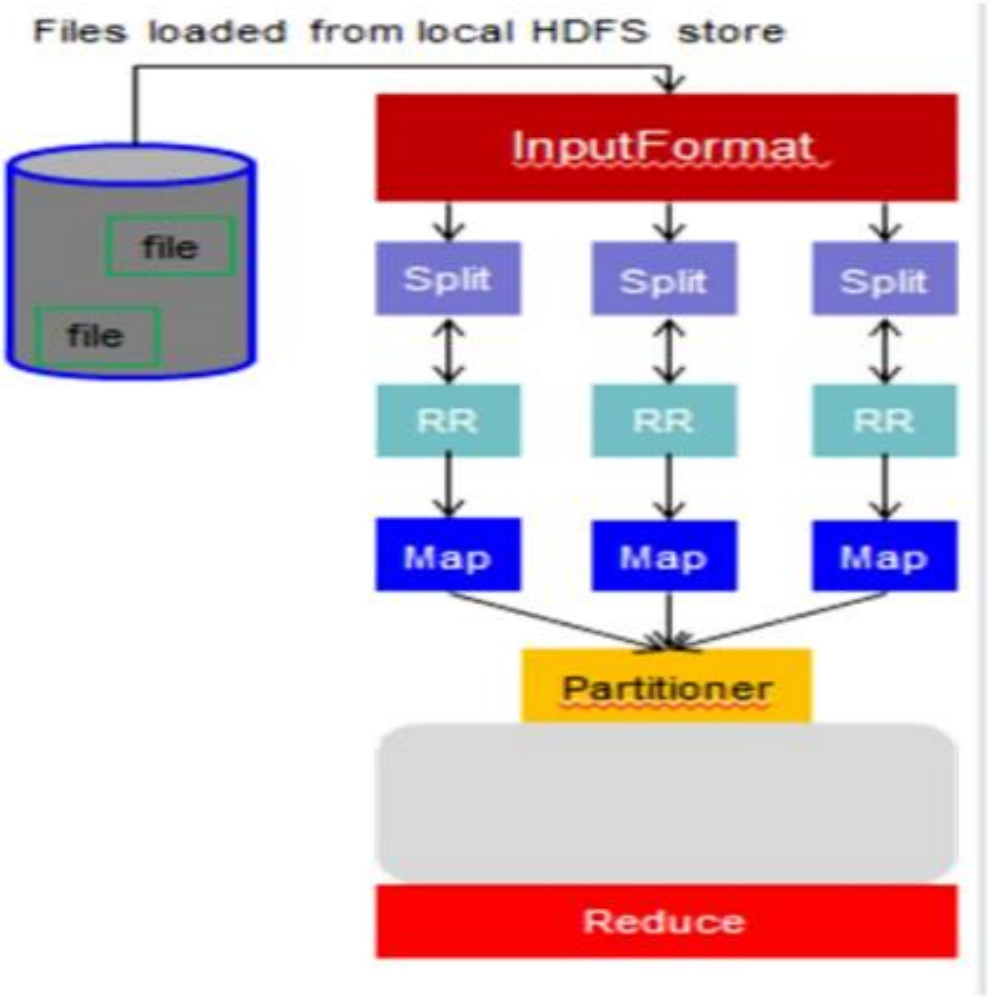
6. MAPPER AND REDUCER :

- The *Mapper* performs the user-defined work of the first phase of the MapReduce program
- A new instance of Mapper is created for each split
- The *Reducer* performs the user-defined work of the second phase of the MapReduce program
- A new instance of Reducer is created for each partition
- *For each key in the partition assigned to a Reducer, the Reducer is called once*



7. PARTITIONER :

- Each mapper may emit (K, V) pairs to *any* partition
- Therefore, the map nodes must all agree on where to send different pieces of intermediate data
- The *partitioner* class determines which partition a given (K,V) pair will go to
- The default partitioner computes *a hash value* for a given key and assigns it to a partition based on this result.

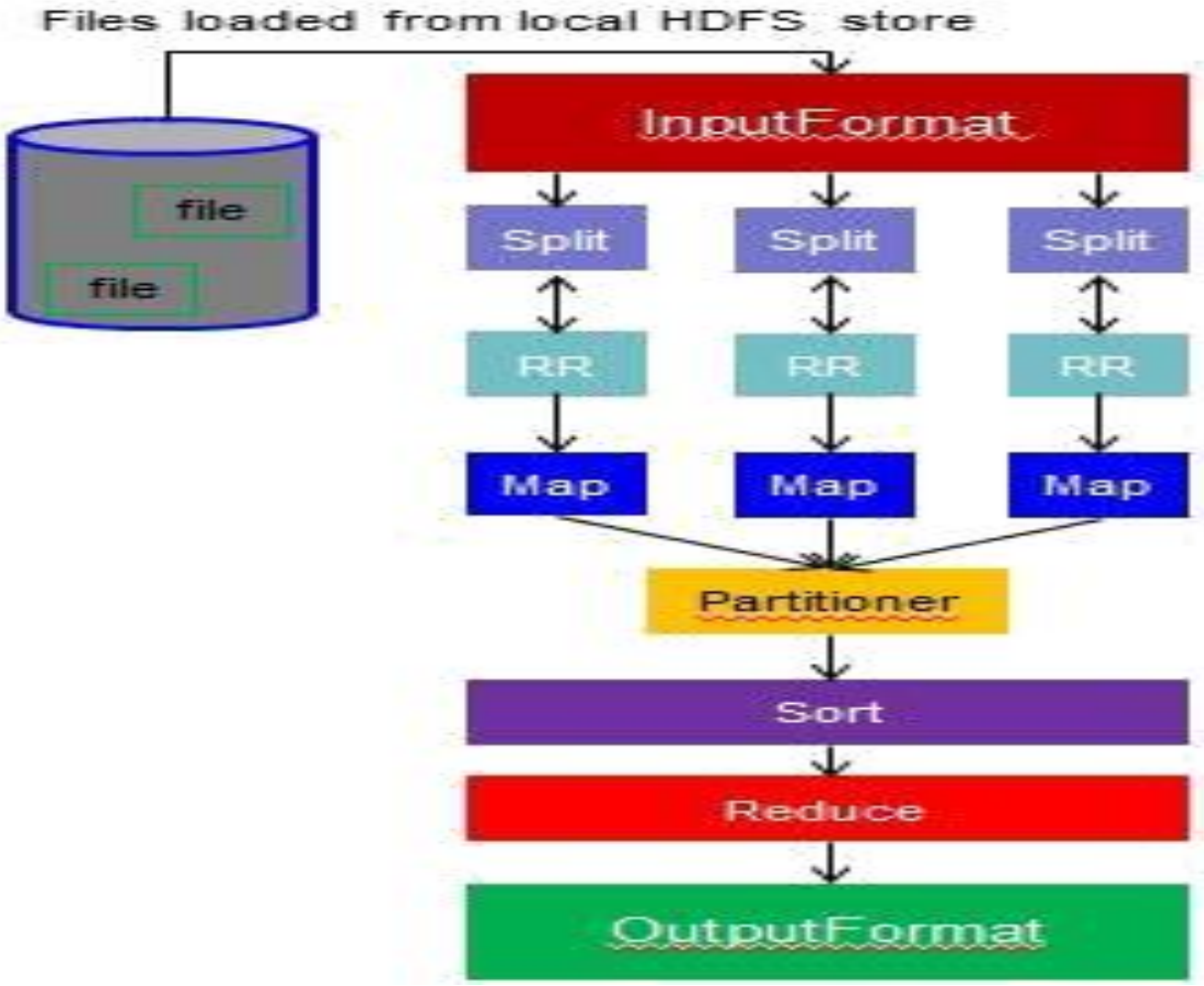


8. SORT :

- Each Reducer is responsible for reducing the values associated with (several) intermediate keys
- The set of intermediate keys on a single node is *automatically sorted* by MapReduce before they are presented to the Reducer.

9. OUTPUT :

- The *OutputFormat* class defines the way (K,V) pairs produced by Reducers are written to output files.
- The instances of OutputFormat provided by Hadoop write to files on the local disk or in HDFS



- **FILE FORMATS:**

  - part – m : map files are stored in this file format
  - part – r : reduce files are stored in this file format.

- **CREATING NAME NODE AND RUNNING JAR FILES:**

1. Starting hadoop daemon

```
stuti@stuti:~$ start-dfs.sh
Starting namenodes on [localhost]
localhost: starting namenode, logging to /usr/local/hadoop/hadoop-2.7.3/logs/had
oop-utsav-namenode-ubuntu.out
localhost: starting datanode, logging to /usr/local/hadoop/hadoop-2.7.3/logs/had
oop-utsav-datanode-ubuntu.out
Starting secondary namenodes [0.0.0.0]
0.0.0.0: starting secondarynamenode, logging to /usr/local/hadoop/hadoop-2.7.3/l
ogs/hadoop-utsav-secondarynamenode-ubuntu.out
stuti@stuti:~$ start-yarn.sh
starting yarn daemons
starting resourcemanager, logging to /usr/local/hadoop/hadoop-2.7.3/logs/yarn-ut
sav-resourcemanager-ubuntu.out
localhost: starting nodemanager, logging to /usr/local/hadoop/hadoop-2.7.3/logs/
yarn-utsav-nodemanager-ubuntu.out
```

2. After starting hadoop daemons

```
stuti@stuti:~$ jps
23521 Jps
23074 ResourceManager
22565 NameNode
18521 org.eclipse.equinox.launcher_1.5.100.v20180827-1352.jar
22716 DataNode
22909 SecondaryNameNode
23198 NodeManager
stuti@stuti:~$
```

3. Creating a name node in hdfs :

```
stuti@stuti:~$ hadoop fs -mkdir /uts1
stuti@stuti:~$ hadoop fs -ls /
Found 1 items
drwxr-xr-x   - utsav supergroup          0 2018-11-13 07:05 /uts1
stuti@stuti:~$
```

4. Copying files from local file system(lfs) to hadoop distributed file system(hdfs)

```
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=124
        File Output Format Counters
                Bytes Written=92
stuti@stuti:/Desktop/utsavProject$ hadoop fs -ls /
Found 3 items
drwx------   - utsav supergroup          0 2018-11-13 07:55 /tmp
drwxr-xr-x   - utsav supergroup          0 2018-11-13 07:56 /user
drwxr-xr-x   - utsav supergroup          0 2018-11-13 07:09 /uts1
stuti@stuti:/Desktop/utsavProject$ hadoop fs -ls /user
Found 1 items
drwxr-xr-x   - utsav supergroup          0 2018-11-13 07:56 /user/utsav
stuti@stuti:/Desktop/utsavProject$ hadoop fs -ls /user/utsav
Found 1 items
drwxr-xr-x   - utsav supergroup          0 2018-11-13 07:56 /user/utsav/OUT1
stuti@stuti:/Desktop/utsavProject$ hadoop fs -ls /user/utsav/OUT1
Found 2 items
-rw-r--r--   4 utsav supergroup          0 2018-11-13 07:56 /user/utsav/OUT1/_SUCCESS
-rw-r--r--   4 utsav supergroup         92 2018-11-13 07:56 /user/utsav/OUT1/part-r-00000
stuti@stuti:/Desktop/utsavProject$
```

5. Running the jar file

```
-rw-r--r--   4 utsav supergroup        124 2018-11-13 07:08 /uts1/para.txt
stuti@stuti:~/Desktop/utsavProject$ hadoop jar word.jar word.word /uts1/para.txt OUT1
18/11/13 07:55:24 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
18/11/13 07:55:25 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement the Tool interface and execu
te your application with ToolRunner to remedy this.
18/11/13 07:55:25 INFO input.FileInputFormat: Total input paths to process : 1
18/11/13 07:55:26 INFO mapreduce.JobSubmitter: number of splits:1
18/11/13 07:55:26 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1542117336257_0001
18/11/13 07:55:33 INFO impl.YarnClientImpl: Submitted application application_1542117336257_0001
18/11/13 07:55:33 INFO mapreduce.Job: The url to track the job: http://ubuntu:8088/proxy/application_1542117336257_0001/
18/11/13 07:55:33 INFO mapreduce.Job: Running job: job_1542117336257_0001
18/11/13 07:56:17 INFO mapreduce.Job: Job job_1542117336257_0001 running in uber mode : false
18/11/13 07:56:17 INFO mapreduce.Job:  map 0% reduce 0%
18/11/13 07:56:50 INFO mapreduce.Job:  map 100% reduce 0%
18/11/13 07:56:59 INFO mapreduce.Job:  map 100% reduce 100%
18/11/13 07:57:00 INFO mapreduce.Job: Job job_1542117336257_0001 completed successfully
18/11/13 07:57:00 INFO mapreduce.Job: Counters: 49
        File System Counters
                FILE: Number of bytes read=146
                FILE: Number of bytes written=237513
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=224
                HDFS: Number of bytes written=92
                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)=30779
                Total time spent by all reduces in occupied slots (ms)=4766
                Total time spent by all map tasks (ms)=30779
                Total time spent by all reduce tasks (ms)=4766
                Total vcore-milliseconds taken by all map tasks=30779
                Total vcore-milliseconds taken by all reduce tasks=4766
                Total megabyte-milliseconds taken by all map tasks=31517696
                Total megabyte-milliseconds taken by all reduce tasks=4880384
        Map-Reduce Framework
                Map input records=6
                Map output records=25
```

```
stuti@stuti:~/Desktop/utsavProject$ stop-dfs.sh
Stopping namenodes on [localhost]
localhost: stopping namenode
localhost: stopping datanode
Stopping secondary namenodes [0.0.0.0]
0.0.0.0: stopping secondarynamenode
stuti@stuti:~/Desktop/utsavProject$ stop-yarn.sh
stopping yarn daemons
stopping resourcemanager
localhost: stopping nodemanager
localhost: nodemanager did not stop gracefully after 5 seconds: killing with kill -9
no proxyserver to stop
stuti@stuti:~/Desktop/utsavProject$ jps
18521 org.eclipse.equinox.launcher_1.5.100.v20180827-1352.jar
22188 Jps
stuti@stuti:~/Desktop/utsavProject$
```

```
Map-Reduce Framework
        Map input records=6
        Map output records=25
        Map output bytes=224
        Map output materialized bytes=146
        Input split bytes=100
        Combine input records=25
        Combine output records=12
        Reduce input groups=12
        Reduce shuffle bytes=146
        Reduce input records=12
        Reduce output records=12
        Spilled Records=24
        Shuffled Maps =1
        Failed Shuffles=0
        Merged Map outputs=1
        GC time elapsed (ms)=965
        CPU time spent (ms)=15640
        Physical memory (bytes) snapshot=444751872
        Virtual memory (bytes) snapshot=3862724608
        Total committed heap usage (bytes)=284164096
Shuffle Errors
        BAD_ID=0
        CONNECTION=0
        IO_ERROR=0
        WRONG_LENGTH=0
```

- **INPUT FILE:**

```
baggu is a good guitar player
baggu is a good singer
baggu is funny
baggu is single
baggu likes biryani
baggu is a good boy
```

```
stuti@stuti:~/Desktop/utsavProject$ hadoop fs -cat /uts1/para.txt
baggu is a good guitar player
baggu is a good singer
baggu is funny
baggu is single
baggu likes biryani
baggu is a good boy
stuti@stuti:~/Desktop/utsavProject$
```

- **RESULTS :**

The word count of each individual word used in the text file:

```
stuti@stuti:~/Desktop/utsavProject$ hadoop fs -ls /user/utsav/OUT1
Found 2 items
-rw-r--r--   4 utsav supergroup          0 2018-11-13 07:56 /user/utsav/OUT1/_SUCCESS
-rw-r--r--   4 utsav supergroup         92 2018-11-13 07:56 /user/utsav/OUT1/part-r-00000
stuti@stuti:~/Desktop/utsavProject$ hadoop fs -cat /user/utsav/OUT1/part-r-00000
a       3
baggu   6
biryani 1
boy     1
funny   1
good    3
guitar  1
is      5
likes   1
player  1
singer  1
single  1
stuti@stuti:~/Desktop/utsavProject$
```

- **JAVA MAP-REDUCE CODES :**

```
package word;
import java.io.IOException; import

java.util.StringTokenizer;


import org.apache.hadoop.conf.Configuration; import
org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable; import
org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job; import
org.apache.hadoop.mapreduce.Mapper; import
org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat; import
org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;



public class word {

        public static class TokenizerMapper
              extends Mapper<Object, Text, Text, IntWritable>{

          private final static IntWritable one = new IntWritable(1); private Text
          word = new Text();                        // creating new Text object


          public void map(Object key, Text value, Context context
                    ) throws IOException, InterruptedException { StringTokenizer itr =
            new StringTokenizer(value.toString() , "
." );

            while (itr.hasMoreTokens()) {
              word.set(itr.nextToken());          // converting string to Text
Format
```

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

public static class IntSumReducer
    extends Reducer<Text,IntWritable,Text,IntWritable> {
  private IntWritable result = new IntWritable();

  public void reduce(Text key, Iterable<IntWritable> values,
                     Context context
                     ) throws IOException, InterruptedException { int
    sum = 0;
    for (IntWritable val : values) {
      sum += val.get();
    }
    result.set(sum);
    context.write(key, result);
  }
}

public static void main(String[] args) throws Exception {
  Configuration conf = new Configuration();
  Job job = Job.getInstance(conf, "word count"); job.setJarByClass(word.class);
  job.setMapperClass(TokenizerMapper.class);
  job.setCombinerClass(IntSumReducer.class);
  job.setReducerClass(IntSumReducer.class);
  job.setOutputKeyClass(Text.class);
  job.setOutputValueClass(IntWritable.class);
  FileInputFormat.addInputPath(job, new Path(args[0]));
  FileOutputFormat.setOutputPath(job, new Path(args[1]));
  System.exit(job.waitForCompletion(true) ? 0 : 1);
}
```

- **CONCLUSION**

The MapReduce programming model has been successfully used at Google for many different purposes. First, the model is easy to use, even for programmers without experience with parallel and distributed systems, since it hides the details of parallelization, fault-tolerance, locality optimization, and load balancing. Second, a large variety of problems are easily expressible as MapReduce computations. For example, MapReduce is used for the

generation of data for Google's production web search service, for sorting, for data mining, for machine learning, and many other systems. Third, we have developed an implementation of MapReduce that scales to large clusters of machines comprising thousands of machines. The implementation makes efficient use of these machine resources and therefore is suitable for use on many of the large computational problems encountered at Google. We have learned several things from this work. First, restricting the programming model makes it easy to parallelize and distribute computations and to make such computations fault-tolerant. Second, network bandwidth is a scarce resource. A number of optimizations in our system are therefore targeted at reducing the amount of data sent across the network: the locality optimization allows us to read data from local disks, and writing a single copy of the intermediate data to local disk saves network bandwidth. Third, redundant execution can be used to reduce the impact of slow machines, and to handle machine failures and data loss.

- **REFERENCES**

- https://en.wikipedia.org/
- http://hadoop.apache.org/
- https://www.google.co.in/
- https://www.ubuntu.com/
- http://adcalves.wordpress.com/20112/12/a-hadoop-primer/
- http://programminggems.wordpress.com/2012/02/20/cloud- foundry-install-error-no-candidate-version-available-for-sun- java6-bin/
- http://www.michael-noll.com/tutorials/running-hadoop-on- ubuntu-linux-single-node-cluster/
- Map-Reduce: Simplified Data Processing on Large Clusters Jeffrey Dean and Sanjay Ghemawat jeff@google.com, sanjay@google.com Google, Inc.
- Oracle In-Database Hadoop: When MapReduce Meets RDBMS Xueyuan Su Computer Science Yale University New Haven, CT 06520 xueyuan.su@yale.edu Garret Swart Oracle Corporation Redwood Shores, CA 94065 garret.swart@oracle.com SIGMOD‘12, May 20–24, 2012, Scottsdale, Arizona, USA. Copyright 2012 ACM 978-1-4503- 1247-9/12/05 ...$10.00.