Data Concepts

27 December 2023 08:47

5. Map Reduce

Getting in touch with Map Reduce Framework

Lecture
- Hadoop Map Reduce paradigm,
- Map and Reduce tasks,
- Map Reduce Execution Framework,
- Map Reduce Daemons
- Anatomy of a Map Reduce Job run

More Map Reduce Concepts

Lecture
- Partitioners and Combiners,
- Input Formats (Input Splits and Records, Text Input, Binary Input, Multiple Inputs),
- Output Formats (Text Output, Binary Output, Multiple Output).
- Distributed Cache

6. Basics of Map Reduce Programming

Lecture
- Hadoop Data Types,
- Java and Map Reduce,
- Map Reduce program structure,
- Map-only program, Reduce-only program,
- Use of combiner and partitioner,
- Counters, Schedulers (Job Scheduling),
- Custom Writables, Compression

Lab-Assignment:
- Execute the train data example.
- Execute the train data example using chained methods.

Map Reduce Streaming

Lecture
- Complex Map Reduce programming,
- Map Reduce streaming,
- Python and Map Reduce,
- Map Reduce on image dataset

MapReduce

Before we proceed make sure all your VM's are working fine and the Big-data setup is in ready working state.

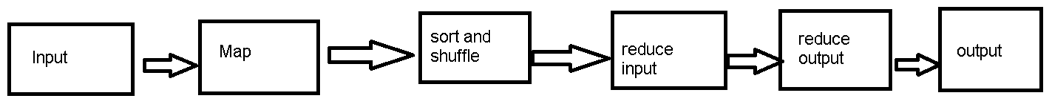
- **MapReduce is a data processing tool which is used to process the data parallelly in a distributed form (typically on Apache Hadoop clusters or on cloud eg: Amazon Elastic MapReduce (EMR) clusters , Azure HDInsight**
- **Developed in 2004 by Google --> based on "MapReduce: Simplified data processing on large clusters"**
- **This is two phases --> The mapper phase and the reducer phase.**
  - **The mapper --> the input is given in the form of a key-value pair & the output of the mapper is fed to the reducer as input**
  - **The reducer --> runs only after the mapper is over, this also takes input in key-value format and the output of reducer is the final output**

Pre-requisite:
- **Java installation -- check whether the java is installed or not --> we can use $ java -version**
- **Hadoop installation --> check whether Hadoop is installed or not --> we can use $hadoop version**

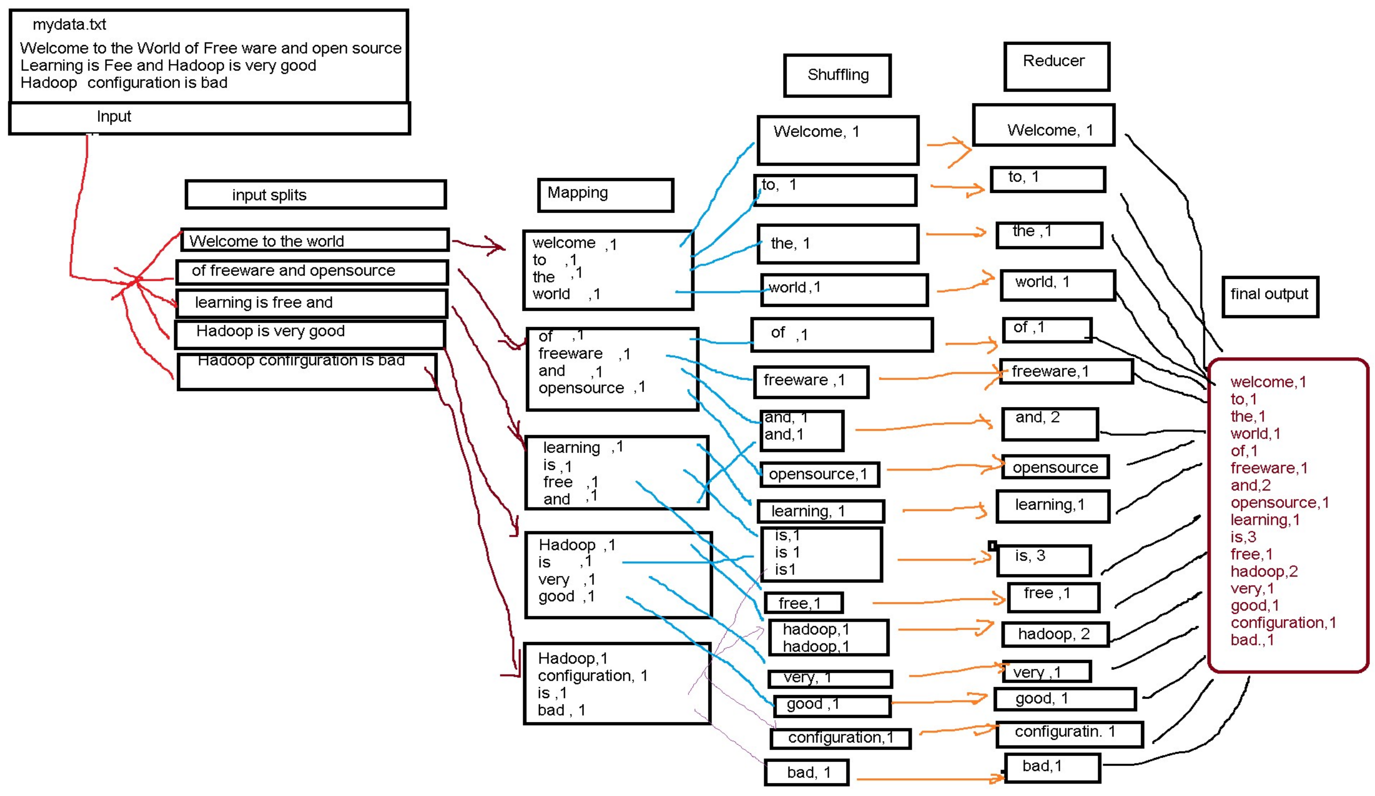
- The only way to access data stored in the HDFS was using MapReduce, but not other query-based methods are utilized. Eg: Hive and pig

Steps in Map Reduce:-

1. **The map takes data in the form of pairs and returns a list of <key,value> pairs. The keys will not be unique**
2. **Using the output of Map, sort and shuffel are applied by the hadoop architecure. This sort and shuffel acts on these list of <key value> pairs and sends out unique keys a list of values associated with this unique k ey < key list (vaues)>**
3. **An output of sort and shuffle is sent to the reducer phase. The reducer perform a defined function on a list values for unique keys and final out <key value> will be stored / displayed.**



Map Reduce Example



The four phases of execution --> splitting, mapping, shuffling and reducing

The data goes through the four phases of MapReduce in Big data

1. **Input Splits:**
   - An input to mapreduce in bigdata is divided into fixed-size called inputs splits
   - Input splits is a chunk of the input that is consumed by a single map
     {in our example we have taken the splits after every four word in the entire text }
2. **Mapping**
   - The 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 out above example, a job of mapping phase is to count a number of occurrences of each word from input splits)
3. **Shuffling**
   - This phase consumes the output of mapping phase. The main task is to consolidate the relevant words from mapping phase output.'
     { in our example, the same words are clubbed together along with their respective frequency }
4. **Recuing**
   - In this phase, output values from the shuffling phases are aggregated. The phase combines values form shuffling phase and returns a single output value. --> This phase summarizes the complete dataset
     { in our example, this phase aggregates the values from shuffling phase i.e, calculates total occurrences of each word }

How MapReduce Organizes work?

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

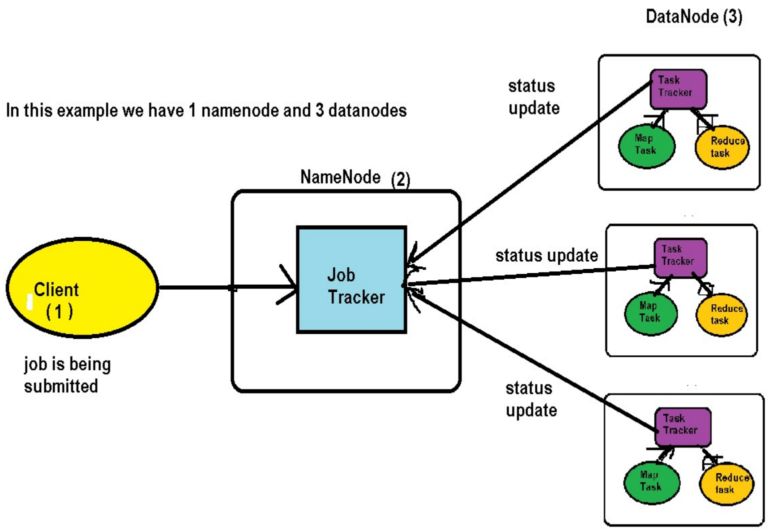
1. **Map tasks (splits and mapping)**
2. **Reduce tasks (shuffling and reducing)**

The complete execution process (exectuion of map and reduce tass, both) are controlled by two type of entities called as

1. **Jobtracker**
2. **Multiple Task Trackers**

Jobtracker --> acts like a master -- responsible for complete execution of submitted job
Multiple task tracker --> acts like a slave --> each slave will performing the job

For every job submitted for execution in the systems, there is one jobtracker that resides on the namenode and there are multiple tasktrackers which resides on datanode.



Hadoop MapReduce Working schematic

- **A job is divided into multiple tasks which are then run onto multiple data nodes in a cluster.**
- **It is the responsibility of job tracker to coordinate the activity by scheduling tasks to run on a different data nodes**
- **Execution of individual task is then to look after by the task tracker, which resides on every data node executing part of the job**
- **Task tracker's responsibility is to send the progress report to the job tracker**
- **In addition, task tracker periodically sends 'heartbeat' signal to the job tracker so as to notify about the current state of the system**
- **Job tracker keeps track of the overall progress of each job, In the event of task failure, the job tracker can reschedule it on a different task tracker.**

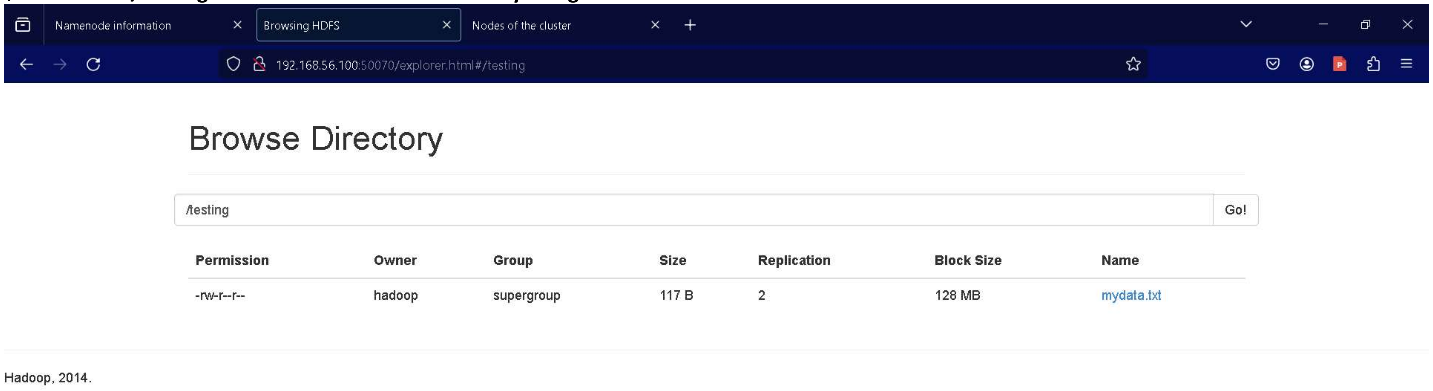
MapReduce Architecture:

- One map task is created for each split which then executes map function for each record in the split.
- It is always beneficial to have multiple splits because the time taken to process a split is small as compared to the time taken for processing of the whole input. When the splits are smaller, the processing is better to load balanced since we are processing the splits in parallel.
- However, it is also not desirable to have splits too small in size. When splits are too small, the overload of managing the splits and map task creation begins to dominate the total job execution time.
- For most jobs, it is better to make a split size equal to the size of an HDFS block (which is 64 MB, by default).
- Execution of map tasks results into writing output to a local disk on the respective node and not to HDFS.
- Reason for choosing local disk over HDFS is, to avoid replication which takes place in case of HDFS store operation.
- Map output is intermediate output which is processed by reduce tasks to produce the final output.
- Once the job is complete, the map output can be thrown away. So, storing it in HDFS with replication becomes overkill.
- In the event of node failure, before the map output is consumed by the reduce task, Hadoop reruns the map task on another node and re-creates the map output.
- Reduce task doesn’t work on the concept of data locality. An output of every map task is fed to the reduce task. Map output is transferred to the machine where reduce task is running.
- On this machine, the output is merged and then passed to the user-defined reduce function.
- Unlike the map output, reduce output is stored in HDFS (the first replica is stored on the local node and other replicas are stored

on off-rack nodes). So, writing the reduce output

MapReduce Word Count Example

```
$ hdfs dfs -mkdir /testing
$ nano mydata.txt
```
Welcome to the world of freeware and opensource
Learning is free and Hadoop is very good
Hadoop configuration is bad
<save and exit>
$ hdfs dfs -put ./mydata.txt /testing
$ hdfs dfs -ls /testing or we can browse the directory using web interface



Browse Directory

/testing Go!

Permission	Owner	Group	Size	Replication	Block Size	Name
-rw-r--r--	hadoop	supergroup	117 B	2	128 MB	mydata.txt

Hadoop, 2014.

Perform the activity on all the nodes

```
$ su
# apt-get install default-jdk
# nano /etc/bash.bashrc
export JAVA_HOME=/usr/lib/jvm/java-11-openjdk-amd64
<save and exit>

# nano /usr/local/hadoop/etc/hadoop/hadoop-env.sh
export JAVA_HOME=/usr/lib/jvm/java-11-openjdk-amd64
<save and exit>
# exit
```

Execute in the mainnode only {by logging as hadoop user)

```
$ stop-yarn.sh
$stop-df.sh
```

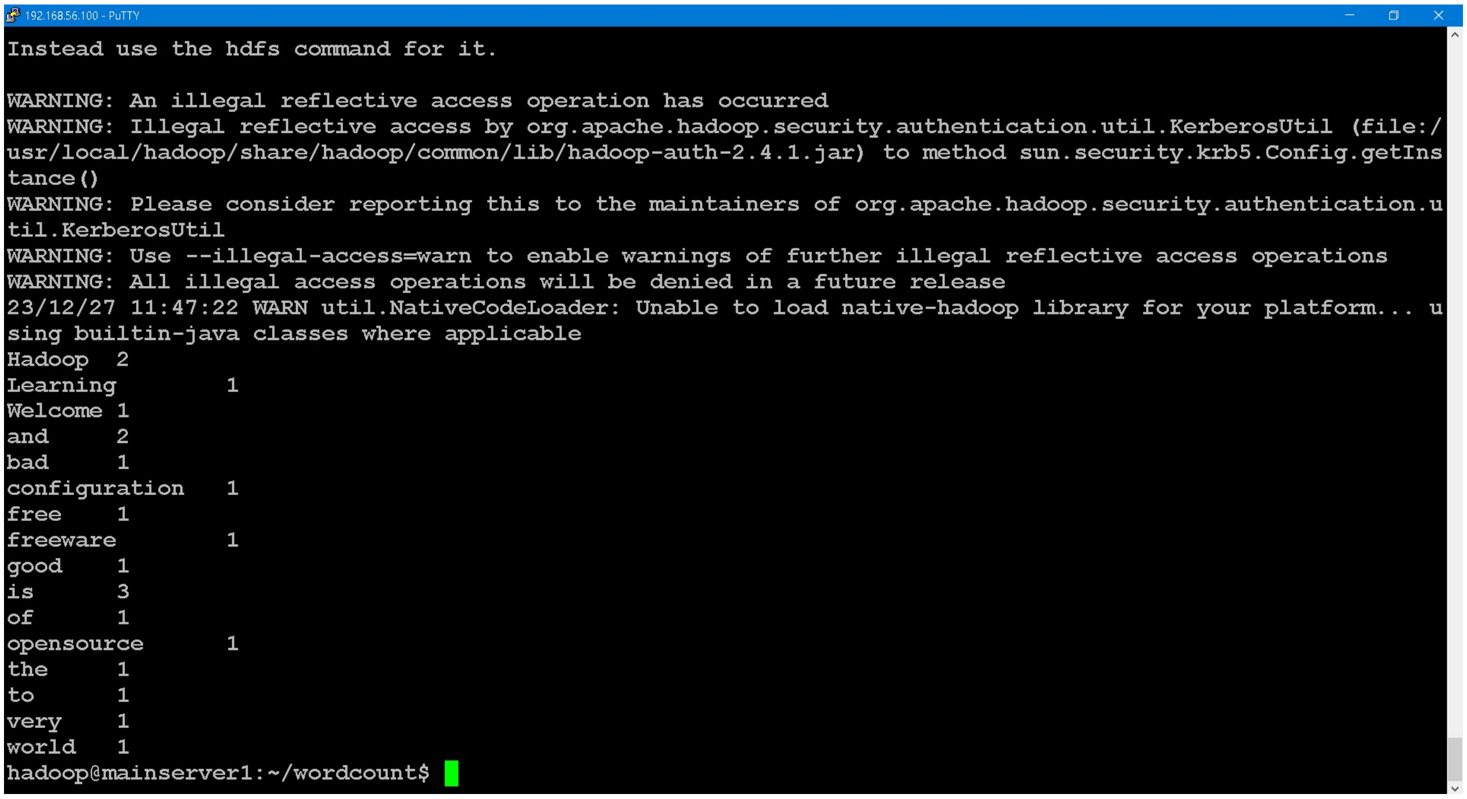
Execute in all the slave nodes to verify
```
$ jps
```

Execute in the mainnode only {by logging as hadoop user)
```
$start-dfs.sh
$start-yarn.sh
```

Execute in all the nodes to verify
```
$ jps
$ ss a-ant
```

Now login to the Namename i.e master server and perform the following activity

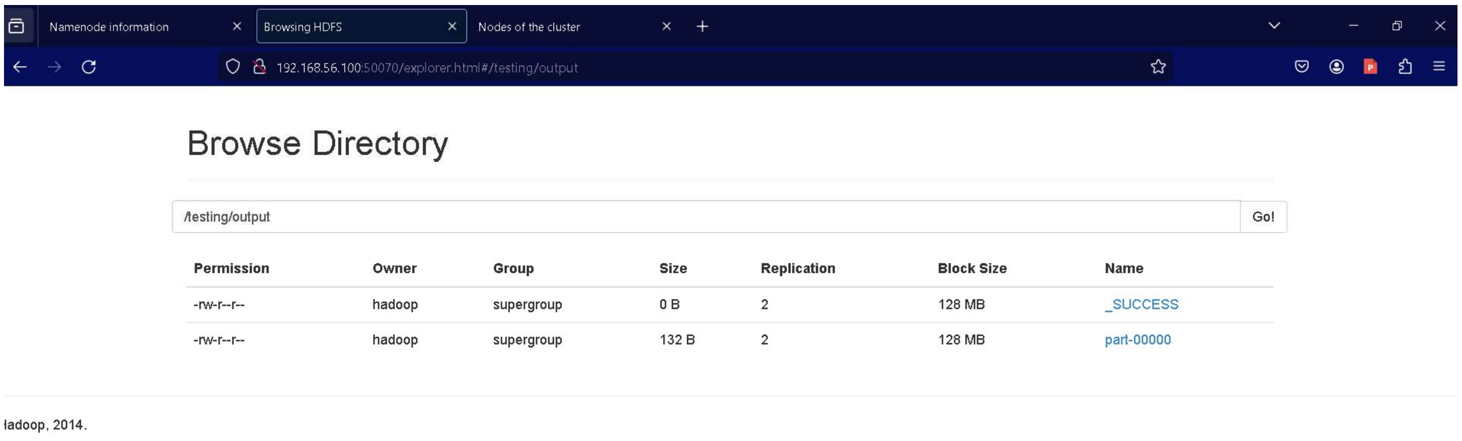
```
$ cd
$ mkdir wordcount
$ ls -l
$ mv  *.java   ./wordcount/
 $ cd wordcount/
 $  ls -l
  $  javac WordCountMapper.java -cp $(hadoop classpath)
 $  javac WordCountReducer.java -cp $(hadoop classpath)
$  javac WordCount.java -cp $(hadoop classpath):/home/hadoop/wordcount
$  ls -l    { you would be able to view 3 java class files }
 $ jar -cf demowordcount.jar *
$   jar -tf demowordcount.jar
 $ hadoop dfs -ls /testing
$ hadoop jar ./demowordcount.jar  WordCount /testing/mydata.txt  /testing/output
 $ hadoop dfs -cat /testing/output/part-00000
```



```
Instead use the hdfs command for it.

WARNING: An illegal reflective access operation has occurred
WARNING: Illegal reflective access by org.apache.hadoop.security.authentication.util.KerberosUtil (file:/
usr/local/hadoop/share/hadoop/common/lib/hadoop-auth-2.4.1.jar) to method sun.security.krb5.Config.getIns
tance()
WARNING: Please consider reporting this to the maintainers of org.apache.hadoop.security.authentication.u
til.KerberosUtil
WARNING: Use --illegal-access=warn to enable warnings of further illegal reflective access operations
WARNING: All illegal access operations will be denied in a future release
23/12/27 11:47:22 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... u
sing builtin-java classes where applicable
Hadoop  2
Learning        1
Welcome 1
and     2
bad     1
configuration   1
free    1
freeware        1
good    1
is      3
of      1
opensource      1
the     1
to      1
very    1
world   1
hadoop@mainserver1:~/wordcount$
```

Final output in your web browser



Browse Directory

/testing/output

Permission	Owner	Group	Size	Replication	Block Size	Name
-rw-r--r--	hadoop	supergroup	0 B	2	128 MB	_SUCCESS
-rw-r--r--	hadoop	supergroup	132 B	2	128 MB	part-00000

ladoop, 2014.