Cluster Computing Assignment

Submitted By: 2644995

UNIVERSITY of STIRLING

Step1:

Introduction to HDFS:

Hadoop is a system for large-scale data processing, and at the core of Hadoop, there are two components. One is the data storage, which is Hadoop, or HDFS. The other is the data processing part, which is called Map Reducer.

A distributed file system that runs well on ample (large) clusters of commodity machines is known as HDFS. It is based on the Google GFS paper, and it provides redundant storage for massive datasets. HDFS comes with its advantages and disadvantages. [Table 1]. HDFS is capable of handling very, very large files and streaming data access. It runs on commodity hardware. HDFS is known for its fault tolerance as well, and it is optimized for MapReduce programming. But it comes with its disadvantages [Table 1]. HDFS is not really good at low-latency data access, which is required for SQL or NoSQL. The way HDFS distributes and manages data may cause handling lots of small files, which may introduce an overhead into the performance.

Advantages	Disadvantages
Handling very large files	Low-latency data access (SQL, NoSQL)
Streaming data access	Handling lots of small files
Commodity hardware	
Fault tolerance	
Optimized for MapReduce programming	

Table 1

Data Storage in HDFS:

When files are added to the HDFS, it will be split into blocks. Default block size is 64 megs, but it may also be 128 megs, 256 megs, one gigabyte, etc.And why does HDFS use blocks? The first reason is replication. HDFS replicates all files in a predefined number of replications. That helps with dealing with fault tolerance. Using blocks helps get large files to be chunked and distributed easily. In addition, blocks help with one of the main strengths of Hadoop's system, which is data-local distributed computation. That is, moving to computation to the data for MapReduce functions. Any file written in HDFS is replicated based on the replication factor, which is three by default. Every block is replicated to nodes throughout the cluster [Figure 1]. Replication increases the reliability and performance. Replication increases the reliability, because the system cannot tolerate data loss.

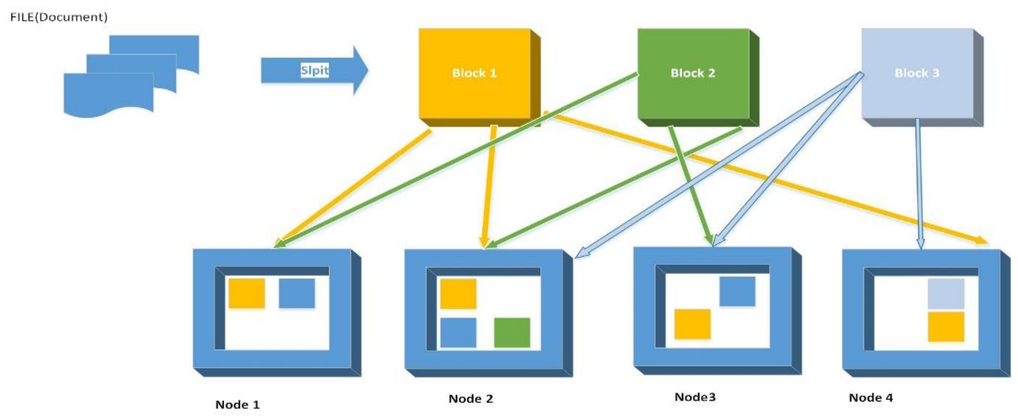


Figure1: Replication of blocks to nodes throughout the cluster

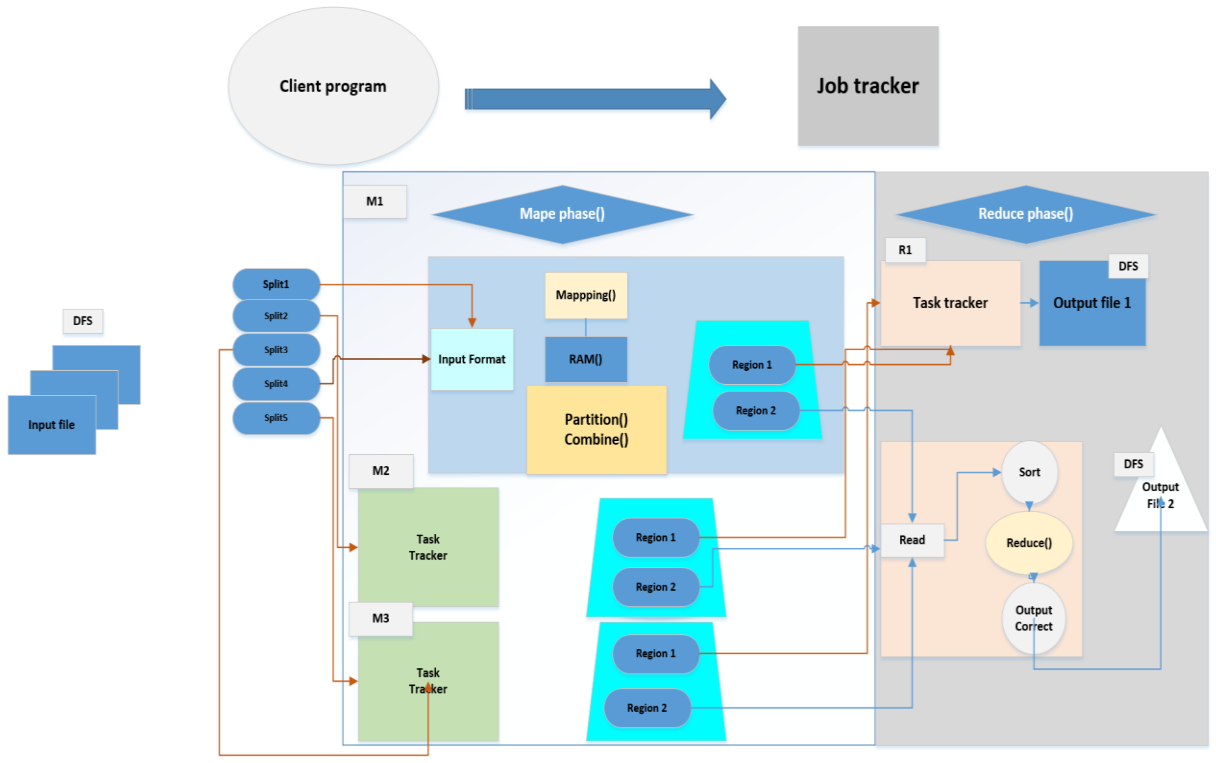


Figure2: The diagrammatic representation of HDFS architecture

HDFS commands used to create a directory:

cd hadoop/Assignment/

hdfs dfs –mkdir Assignment/pouria

```
hdfs dfs –mkdir Assignment/pouria/cache
hdfs dfs –copyFromLocal ratedReviews.txt /user/smo/Assignment/pouria/cache
```

<u>HDFS has 5 different nodes which are outlined as follows:</u>

- **NameNode:**
  Name node saves only metadata of the files, NameNode never store the files. NameNode supervises coordinates access to the data saved in DataNode and the health of Data Node. Namenode watches for all the file system linked information such as:
  i. In which part of the cluster, file is saved.
  ii. Preceding access time for the files.
  -file accessibility for User means permissions like who can access the file.
- **DataNode:**
  DataNode manages interrelates with the blocks and the state of an HDFS node. A DataNode is able to do CPU jobs like machine learning, statistics ,clustering, data import, data export and ... A DataNode needs lot of I/O for transfer and data processing.
- **JobTracker:**
  JobTracker coordinates the parallel processing of data using MapReduce. Firstly, Client submit jobs to the jobtracker. Then jobtracker asks namenode for the confirmation of the data location. After that jobtracker assigns task to the available task tracker. Then jobtracker submit the jar file of the job to the tasktracker. After completion of the task, jobtracker updates its status.
- **Secondary Name Node:**
  This name is little bit confusing. Secondary Node is NOT high availability node for Name node or the backup. Work of Secondary name node is to take image copy of the name node in every 1 hr. Name node stores his metadata in RAM so if namenode goes down you lose everything from the RAM. Hence, after namenode crashes, secondary name node helps the system to recover name node again using its backup.
- **TaskTracker:** In cluster this node accepts tasks from jobtracker like Map, Reduce shuffle etc. Every single tasktracker form the cluster is configured with the set of slots that represents how many tasks it can accept. Tasktracker needs a separate JVM

processes for the task. Because of this, task tracker never goes down even after failure. After completion of the process, task tracker notifies the status of the jobs to the job tracker.

Step 2:

- **Design one:**

Using Map Reducer-

This approach will undergo two phases: ‘map’ and ‘reduce’ and for the problem in question, only one reducer function is required because it is related to finding most common words in each rate review. To achieve this, *IntSumReducer* class has been used. In this approach, the key and the value is used (k:v) and is well suited when the data available, is in small amount. The key is the unique number, in this case, it is rate review (value of review lies between 1 to 9 and value is the word in each rated review. To get the number of occurrences of similar words, a counter [Integer Counter =Entry.getvalue();] is created. <Text, Text, Text, Text> are passed to the reducer using public static class *IntSumReducer* class such that integer value could parse through it easily. After splitting, the job of the mapper is to map the data line by line and then transfer it (mapped data) to the reducer and the reducer is going to process it (e.g. counting the most common word for each rate review).

- **Design Two:**

Using Mapper Combiner Reducer:

This approach is very helpful in reducing network congestion. It enhances the overall performance of the reducer by minimizing the volume of data transfer between Map Class & Reduce Class (the combiner class i.e. *IntSumCombiner* is used between these two classes to achieve it). This approach is well suited when there is large amount of data and can produce a summary from a large dataset. There is no predefined interface for combiner, thus it needs to implement the Reducer Interface’s reduce() method. To get the output, the key-value along a different counter needs to be used. Along with a dictionary, an array is used and two reducer functions are written using this approach. Furthermore, Combiner must have the same key-value types (for input/output) as the reducer class because it operates on each map output key.

__**The mechanism is illustrated as:**__

(Input) `<key1, value1>` -> **map** -> `<key2, value2>` -> **combine** -> `<key2, value2>` -> **reduce** -> `<key3, value3>` (output).

Key and values to be emitted by the mapper are:

The key which is going to be emitted by the mapper is 'Rate of the Review' and the value to be emitted by the mapper is Words from .txt file.

Amount of data that moved across the network in case of both designs:

- In case of design one i.e. using Mapper Reducer, as the single reducer is used, so the amount of data moved across the network is more which leads to enormous network congestion because when a MapReduce job is run, large chunks of intermediate data is generated by the mapper and passed to the Reducer by the framework for further processing.

- In case of design two i.e. using Mapper Combiner Reducer, though the amount of data to be moved across the network is high but the time taken for data transfer (between mapper & reducer) is reduced by the combiner which reduces the amount of data to be processed by the reducer i.e. Combiner acts as a mini-reducer in this design approach.

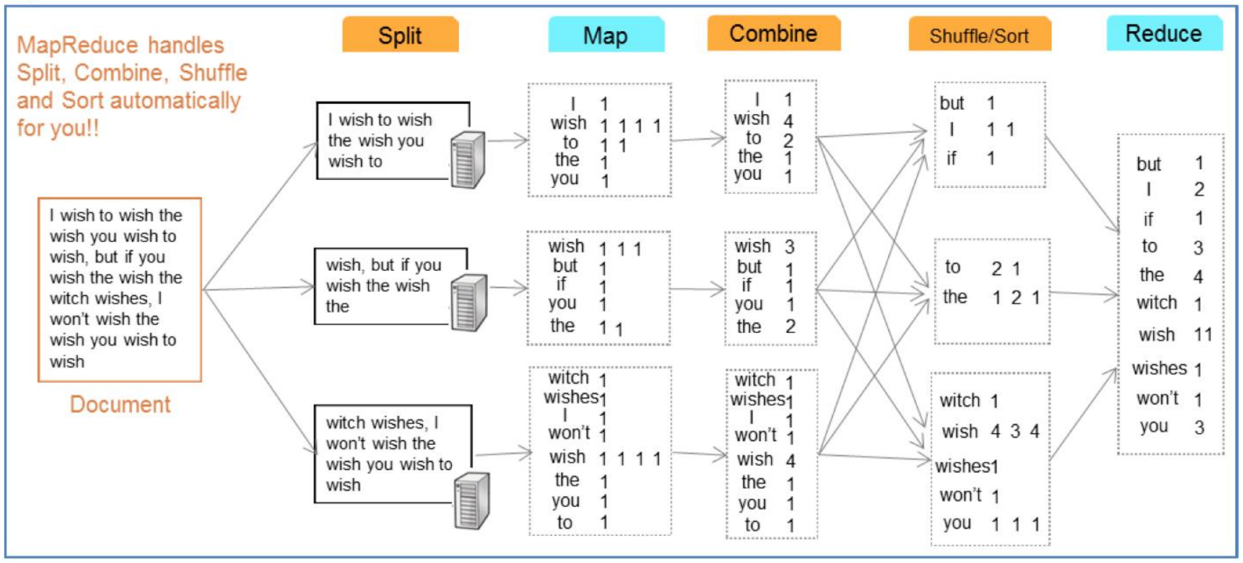

MapReduce handles Split, Combine, Shuffle and Sort automatically for you!!
Split
Map
Combine
Shuffle/Sort
Reduce
I wish to wish the wish you wish to wish, but if you wish the wish the witch wishes, I won't wish the wish you wish to wish
Document
I wish to wish the wish you wish to
wish, but if you wish the wish the
witch wishes, I won't wish the wish you wish to wish
I 1 wish 1 1 1 1 to 1 1 the 1 you 1
wish 1 1 1 but 1 if 1 you 1 the 1 1
witch 1 wishes 1 I 1 won't 1 wish 1 1 1 1 the 1 you 1 to 1
I 1 wish 4 to 2 the 1 you 1
wish 3 but 1 if 1 you 1 the 2
witch 1 wishes 1 I 1 won't 1 wish 4 the 1 you 1 to 1
but 1 I 1 1 if 1
to 2 1 the 1 2 1
witch 1 wish 4 3 4 wishes 1 won't 1 you 1 1 1
but 1 I 2 if 1 to 3 the 4 witch 1 wish 11 wishes 1 won't 1 you 3

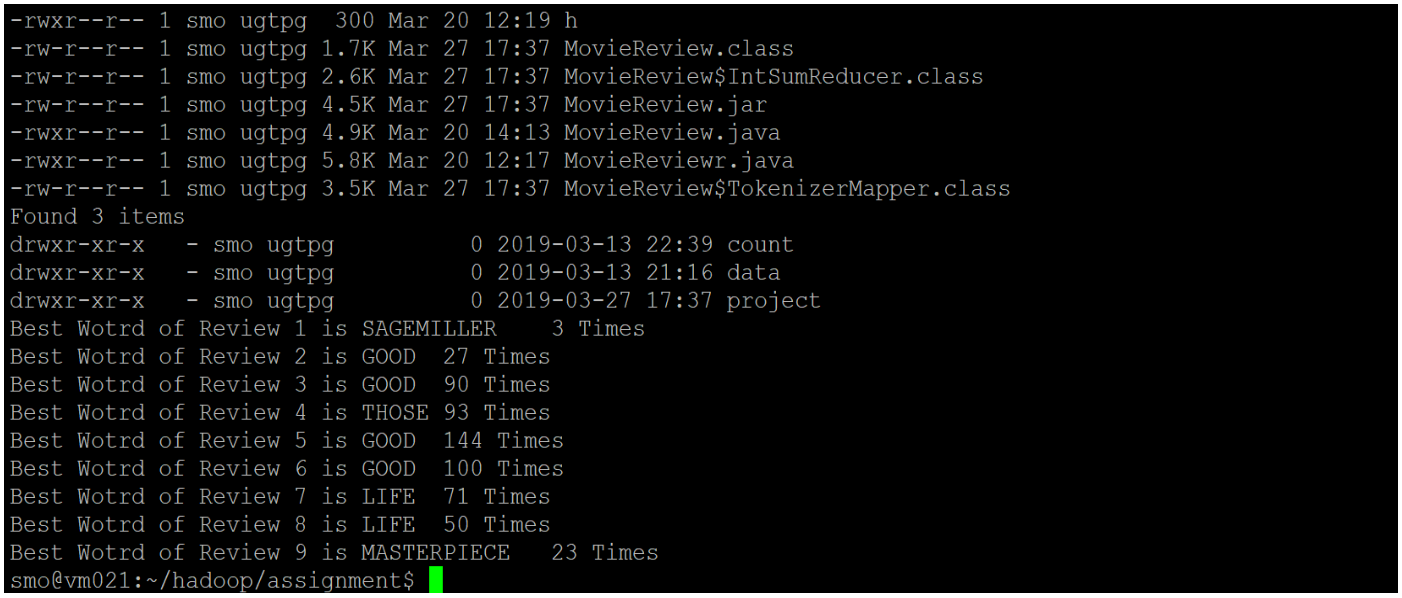
Figure3: Working of design one and design two as explained above

Choice of design for implementation and its Justification:

For the purpose of problem in question, the design one i.e. Map Reducer has been used as the data provided is in a small amount and to fetch out the total number of occurrences, only a key value and counter is required. Reducer has been run for 9 times to get the rating number, the common word including frequency of occurrence of that word. Also, processing time is reduced because of parallel processing of the data by all the nodes (with respect to their part of data).

Step3:

3. RESULT OUTPUT



```
-rwxr--r-- 1 smo ugtpg  300 Mar 20 12:19 h
-rw-r--r-- 1 smo ugtpg 1.7K Mar 27 17:37 MovieReview.class
-rw-r--r-- 1 smo ugtpg 2.6K Mar 27 17:37 MovieReview$IntSumReducer.class
-rw-r--r-- 1 smo ugtpg 4.5K Mar 27 17:37 MovieReview.jar
-rwxr--r-- 1 smo ugtpg 4.9K Mar 20 14:13 MovieReview.java
-rwxr--r-- 1 smo ugtpg 5.8K Mar 20 12:17 MovieReviewr.java
-rw-r--r-- 1 smo ugtpg 3.5K Mar 27 17:37 MovieReview$TokenizerMapper.class
Found 3 items
drwxr-xr-x   - smo ugtpg          0 2019-03-13 22:39 count
drwxr-xr-x   - smo ugtpg          0 2019-03-13 21:16 data
drwxr-xr-x   - smo ugtpg          0 2019-03-27 17:37 project
Best Wotrd of Review 1 is SAGEMILLER    3 Times
Best Wotrd of Review 2 is GOOD   27 Times
Best Wotrd of Review 3 is GOOD   90 Times
Best Wotrd of Review 4 is THOSE 93 Times
Best Wotrd of Review 5 is GOOD   144 Times
Best Wotrd of Review 6 is GOOD   100 Times
Best Wotrd of Review 7 is LIFE   71 Times
Best Wotrd of Review 8 is LIFE   50 Times
Best Wotrd of Review 9 is MASTERPIECE   23 Times
smo@vm021:~/hadoop/assignment$
```