Hadoop Lap Report

1. MapReduce

Hadoop MapReduce is a software framework for easily writing applications which process vast amounts of data in-parallel on large clusters of commodity hardware in a reliable, fault tolerant manner [1]. A typical MapReduce computation processes many terabytes of data on thousands of machines. It is an application running on Yarn and is one of the two frameworks Hadoop is based on, the other being the Hadoop Distributed File System (HDFS), both of which have a master/slave architecture.

MapReduce uses a divide/aggregate approach to perform distributed data processing, usually aiming to have the processing operations as close as possible to where the data resides, processing operations are generally carried out on the same node or rack as the data in order to reduce traffic on the main network. Moreover, as the name suggest, MapReduce computation is based on two interfaces: Mapper and Reducer.

The Mapper performs the *Map* job which takes a set of data, splits it up, and translates the individual elements into a set of intermediate key/value pairs known as tuples. The data set generated by the Map function is then passed onto the Reducer interface which consists of three primary phases: Shuffle, Sort, and Reduce.

The Reducer first shuffles the data set passed to it by fetching the relevant partition of the output of all the mappers. These are then sorted by grouping all identical intermediate keys and sending them to a single node to be processed by the Reduce function. The Reducing phase then merges all intermediate values associated with the same intermediate key to form and generate a possibly smaller data set. These processes are done in parallel on a large cluster of commodity machines and as the data is initially split, failures are independent hence the fault tolerance aspect of the framework. An step by step illustration of MapReduce is shown in the figure below.

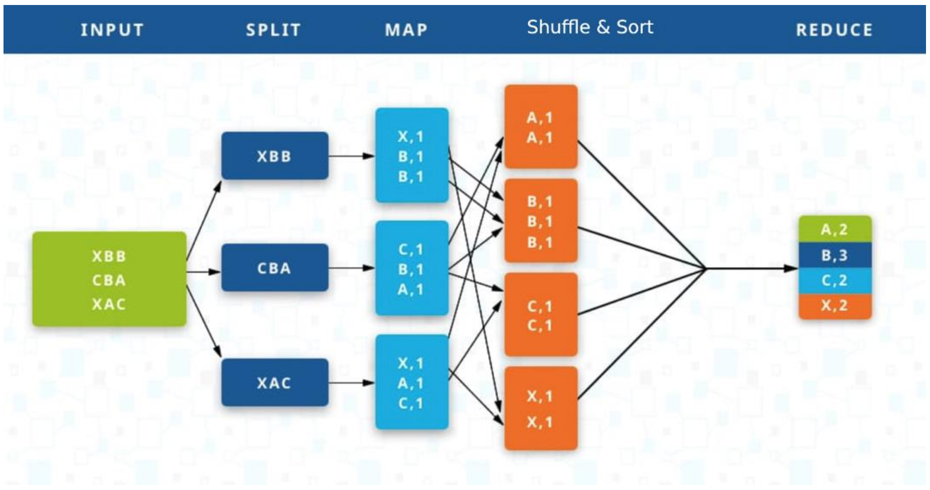


Figure 1: MapReduce step by step illustration. [2]

One of the most important advantages of MapReduce, apart from the capability of computing Big Data is that the run-time system takes care of the details of partitioning the input data, scheduling the program's execution across the commodity 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 [3].

Furthermore, a MapReduce engine consists of a Job Tracker (master node) and multiple Task Trackers (slave nodes). The Job Tracker is where MapReduce jobs are submitted by the client applications. The Job Tracker then determines the location of the data to be processed by a query to the NameNode, which is the centrepiece of the HDFS responsible for tracking where across the cluster the file data is kept as well as keeping the directory tree of all files in the file system [4]. Using the data location, the Job Tracker then decides on which Task Tracker node is best to push the work to without adding much traffic to the main network (by aiming to keep it as close as possible to its data). This is illustrated in the figure below.

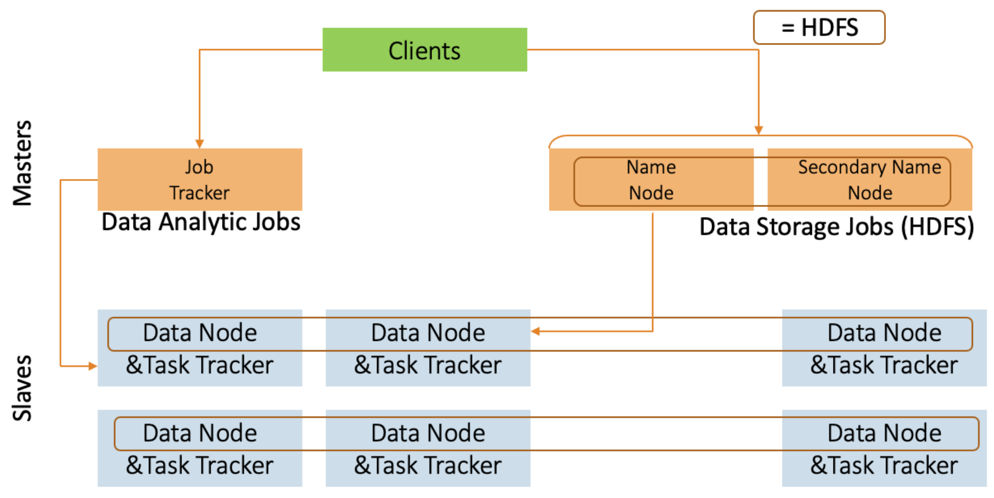
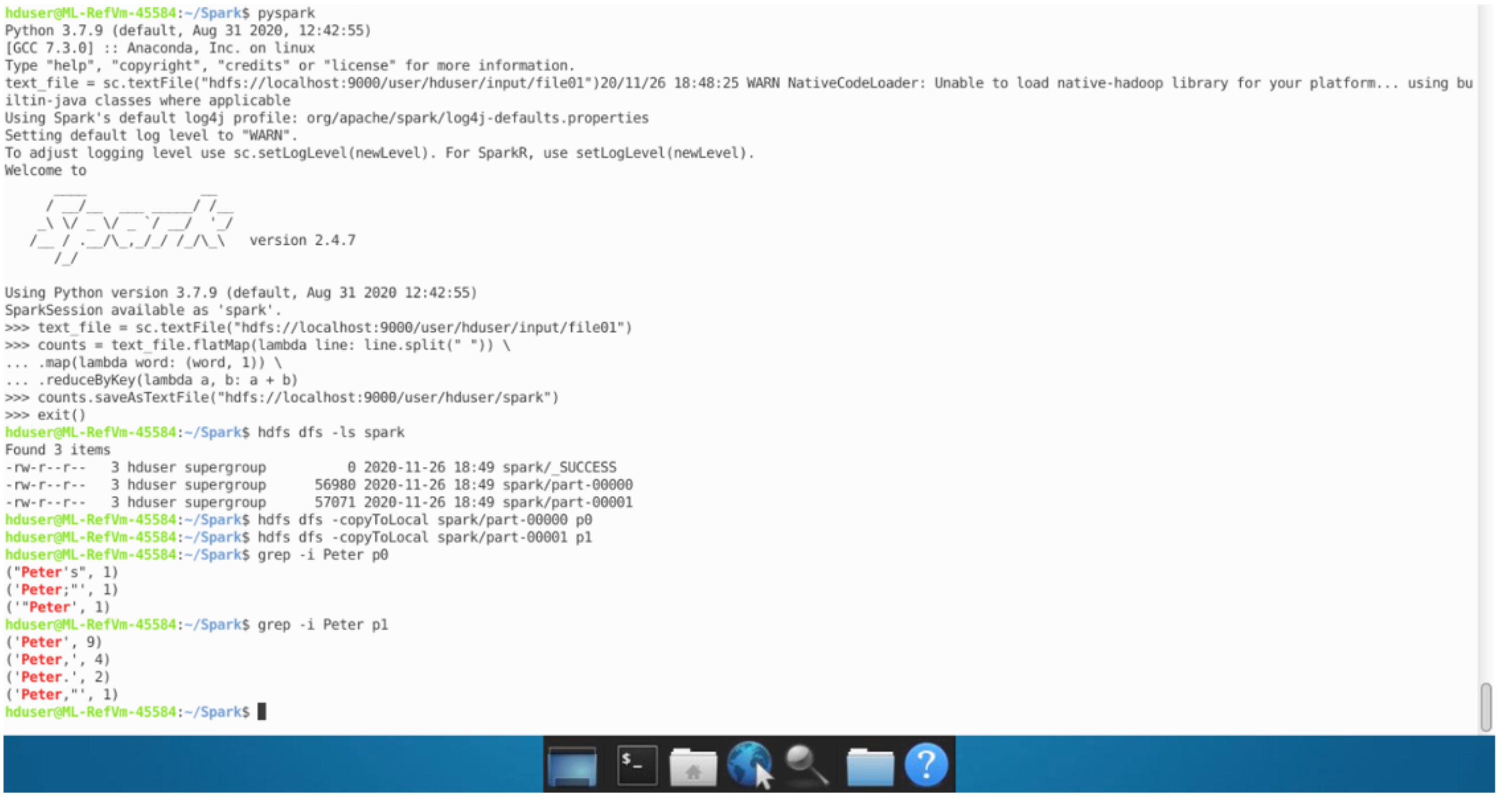


Figure 2: Illustration of Job Tracker, Task Trackers and the architecture of the framework. [5]

The Task Tracker chosen then accepts the tasks pushed down to it and is monitored by sending out regular heartbeat signals to the Job Tracker for the duration of the task. The Task Tracker runs on a Java Virtual Machine (JVM), creating a new JVM instance for each job it does, thus preventing failure of the entire node should one job fail. The Task Tracker monitors all JVM instances capturing outputs and errors, and then informs the Job Tracker of the outcome of the task, upon failure of a task or loss of a heartbeat, the Job Tracker decides on where to reschedule the task to, and in some cases may even blacklist the Task Tracker by marking it as unreliable.

2. Spark



```
hduser@ML-RefVm-45584:~/Spark$ pyspark
Python 3.7.9 (default, Aug 31 2020, 12:42:55)
[GCC 7.3.0] :: Anaconda, Inc. on linux
Type "help", "copyright", "credits" or "license" for more information.
text_file = sc.textFile("hdfs://localhost:9000/user/hduser/input/file01")20/11/26 18:48:25 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using bu
iltin-java classes where applicable
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 2.4.7
      /_/

Using Python version 3.7.9 (default, Aug 31 2020 12:42:55)
SparkSession available as 'spark'.
>>> text_file = sc.textFile("hdfs://localhost:9000/user/hduser/input/file01")
>>> counts = text_file.flatMap(lambda line: line.split(" ")) \
... .map(lambda word: (word, 1)) \
... .reduceByKey(lambda a, b: a + b)
>>> counts.saveAsTextFile("hdfs://localhost:9000/user/hduser/spark")
>>> exit()
hduser@ML-RefVm-45584:~/Spark$ hdfs dfs -ls spark
Found 3 items
-rw-r--r--   3 hduser supergroup          0 2020-11-26 18:49 spark/_SUCCESS
-rw-r--r--   3 hduser supergroup      56980 2020-11-26 18:49 spark/part-00000
-rw-r--r--   3 hduser supergroup      57071 2020-11-26 18:49 spark/part-00001
hduser@ML-RefVm-45584:~/Spark$ hdfs dfs -copyToLocal spark/part-00000 p0
hduser@ML-RefVm-45584:~/Spark$ hdfs dfs -copyToLocal spark/part-00001 p1
hduser@ML-RefVm-45584:~/Spark$ grep -i Peter p0
("Peter's", 1)
('Peter;"', 1)
('"Peter', 1)
hduser@ML-RefVm-45584:~/Spark$ grep -i Peter p1
('Peter', 9)
('Peter,', 4)
('Peter.', 2)
('Peter,"', 1)
hduser@ML-RefVm-45584:~/Spark$ 
```

Figure 3: Screenshot of final exercise output.

The keyword **lambda** is used to create small anonymous functions that can be used wherever function objects are required. They are syntactically restricted to a single expression. In essence, they are just ***syntactic sugar***[1] for a normal function definition [6]. This explains why the declared variable *line* was able to call the built in string function, **split()**, within Python extracting each word from the string into a list or array using the spaces (“ ”) separating them. This list is then passed as a parameter to **flatMap()** function which is a transformation operation. For each element in the Resilient Distributed Dataset (RDD), the **flatMap()** function performs a logic operation specified by the developer and produces zero, one, or more elements. **flatMap()** transforms an RDD of length N into another RDD of length M.
Similarly, the function **map()** is also a transformation operation which performs a specified logic operation on each element of the RDD, however, producing an output RDD with the same number of records as the input. **Map()** transforms an RDD of length N into another RDD of length N. The parameter (word, 1) passed to this function essentially maps every word to a value of 1 so that similar words can be summed.
Lastly, the **reduceByKey()** goes over the mapped words summing the occurrences of unique words using the key, which in this case is the word, finally producing an RDD with the count of every unique word, this is achieved by combining the 1’s written in the previous step. The **reduceByKey()** function views the map of (word, 1) as a sequence of (Key, Value) pairs to be aggregated.

[1] Syntactic sugar is syntax within a programming language that is designed to make things easier to read or to express.

3. References

[1] The Apache Software Foundation, “MapReduce Tutorial,” [Online]. Available: https://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html. [Accessed 7 12 2020].

[2] Talend, “What is MapReduce?,” [Online]. Available: https://www.talend.com/resources/what-is-mapreduce/. [Accessed 7 12 2020].

[3] J. Dean and S. Ghemawet, “MapReduce: Simplified Data Processing on Large Clusters,” Google, Inc., 2004.

[4] ASF Infrabot, “JobTracker,” Confluence, 9 July 2019. [Online]. Available: https://cwiki.apache.org/confluence/display/HADOOP2/JobTracker. [Accessed 7 December 2020].

[5] J. Bonnyman, *Big Data Lecture 1,* University of Nottingham, 2020.

[6] Python, “Chaper 4.7: More on Defining Functions,” [Online]. Available: https://docs.python.org/3/tutorial/controlflow.html. [Accessed 1 December 2020].