Indian Institute of Technology Madras
CS6847
Cloud Computing

The Map Reduce Paradigm

Om Shri Prasath - EE17B113
Pruthvi Raj R G - EE17B114

9 January 2020

Contents

1. Overview

In this assignment we get familiar with the paradigm of map-reduce programming in Hadoop. Map-reduce is a software framework and programming model used for processing huge amounts of data. Map-reduce program work in two phases, namely, Map and Reduce. Map tasks deal with splitting and mapping of data while Reduce tasks shuffle and reduce the data. Hadoop is capable of running MapReduce programs written in various languages: Java, Ruby, Python, and C++. MapReduce programs are parallel in nature, thus are very useful for performing large-scale data analysis using multiple machines in the cluster. The input to each phase is key-value pairs. In addition, every programmer needs to specify two functions: map function and reduce function.

In this assignment we create a cluster of 2 nodes – **One Master(also capable of acting as a data node) and One Slave**. The objective of the assignment is to understand and analyze the Map-Reduce programming model using the New York Taxi dataset(link)
Below we describe the procedure for setting up the HDFS system and we also highlight some of the issue we faces while setting up the system – and the solutions we found out for them.

1. Download the .vdi file from the source.
2. Install VirtalBox in all the nodes. Run the .vdi file to boot up the virtual machine.
3. Follow the instructions given in the BOOS_MOOL_hadoop.pdf
4. Before starting the hdfs we need to make sure that the **folders(namenode and datanode) are removed** from the all the slave nodes.
5. By deafult the slave nodes point to **localhost:9000**. Change it to **master:9000** in the coresite.xml file in the hadoop directory.
6. Allocate sufficient RAM and memory to avoid memory and speed issues. Use replication factor as 1 if memory is a constraint.
7. Use **Filezilla** to transfer files to the vm.
8. Write the map reduce codes and test it out on the vm instead of hdfs.
9. Include the line **'!/usr/bin/python'** in all the map reduce codes to make sure the program understands it as Python2 code.
10. Give access to all the map-reduce codes using **chmod**.
11. Leverage the Hadoop **Streaming API** for helping us passing data between our Map and Reduce code via STDIN and STDOUT.
12. Execute the map-reduce programs with varying number of reducers and different values of the slow start parameter.

2. The Map-Reduce Code

The data contains record of multiple taxi travels for different years . We chose to do our analysis on the data of the year 2011. The data consists of key(The date and time of travel), sometimes appended with a random integer to avoid mixing up if the keys have same date and time. We preprocess the data using GoogleColab as the data was very heavy for our laptops to handle. We notice that there are 1 million rows in which the location(longitude and latitude) is 0(missing). We remove all these rows. Later during the assignment we realized that running the map-reduce task was taking more than 40 minutes for each problem statement and sometimes it even caused memory issues while running. Hence we decided to reduce the size of the data to cut down the memory and the time of excecution. We reduced the size of the data from 54 million rows(after removing zeros) to 10 million rows using random sampling.

```
df_sampled.head()
```

	key	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
	2009-06-15 17:26:21.0000001	4.5	2009-06-15 17:26:21 UTC	-73.844311	40.721319	-73.841610	40.712278	1
	2011-06-12 13:33:00.000000128	15.3	2011-06-12 13:33:00 UTC	-73.949477	40.768018	-74.003118	40.733022	5
	2014-07-27 08:22:00.00000022	16.5	2014-07-27 08:22:00 UTC	-73.954850	40.765420	-74.007510	40.738770	2
	2013-07-31 06:51:13.0000001	19.0	2013-07-31 06:51:13 UTC	-73.988834	40.764280	-73.941405	40.838664	1
	2010-05-22 23:29:57.0000003	25.3	2010-05-22 23:29:57 UTC	-74.001343	40.741320	-73.928619	40.864267	1

2.1 Code Explanation

Note : While generating key-value pair, we check that the year is 2011 in the mapper, only if the year is 2011 do we write the key-value pair to output.

2.1.1 Most Popular Route

The key pair being generated is : Month:Pickup Longitude:Pickup-Latitude:Drop-Longitude:Drop-Latitude and the value is number of Passengers (indicating the popularity). This key-value pair is sent to the reducer in sorted order. We will work with a particular route and we keep increasing the count of passengers, until we reach a point where the key changes, after which we will update the current route to the corresponding month by checking whether the present route count is greater than the least number in the list containing the top 5 routes (initially initialized as 0) and then restart the process for the new route sent to the reducer. This keeps on repeating until the last set of routes, at the moment which we will separately check whether it is greater than the given set of top 5 values and then the output is written to the file.

2.1.2 Most Visited Pickup

The key pair being generated is : Month:Pickup Longitude:Pickup-Latitude and the value is number of Passengers (indicating the popularity). This key-value pair is sent to the reducer in sorted order. We will work with a particular pickup and we keep increasing the count of passengers, until we reach a point where the key changes, after which we will update the current pickup location details to the corresponding month by checking whether the present pickup location count is greater than the least number in the list containing the top 5 pickup location (initially initialized as 0) and then restart the process for the new pickup location. This keeps on repeating until the last set of pickup location, at the moment which we will separately check whether it is greater than the given set of top 5 values and then the output is written written to the file.

2.1.3 Most Visited Drops

The key pair being generated is : Month:Drop-Longitude:Drop-Latitude and the value is number of Passengers (indicating the popularity). This key-value pair is sent to the reducer in sorted order. We will work with a particular drop and we keep increasing the count of passengers, until we reach a point where the key changes, after which we will update the current drop location details to the corresponding month by checking whether the present drop location count is greater than the least number in the list containing the top 5 drop location (initially initialized as 0) and then restart the process for the new drop location. This keeps on repeating until the last set of drop location, at the moment which we will separately check whether it is greater than the given set of top 5 values and then the output is written to the file.

2.1.4 Most Visited Drops

The key pair being generated is : Month:Drop-Longitude:Drop-Latitude and the value is number of Passengers (indicating the popularity). This key-value pair is sent to the reducer in sorted order. We will work with a particular drop and we keep increasing the count of passengers, until we reach a point where the key changes, after which we will update the current drop location details to the corresponding month by checking whether the present drop location count is greater than the least number in the list containing the top 5 drop location (initially initialized as 0) and then restart the process for the new drop location. This keeps on repeating until the last set of drop location, at the moment which we will separately check whether it is greater than the given set of top 5 values and then the output is written to the file.

2.1.5 Most Night Life

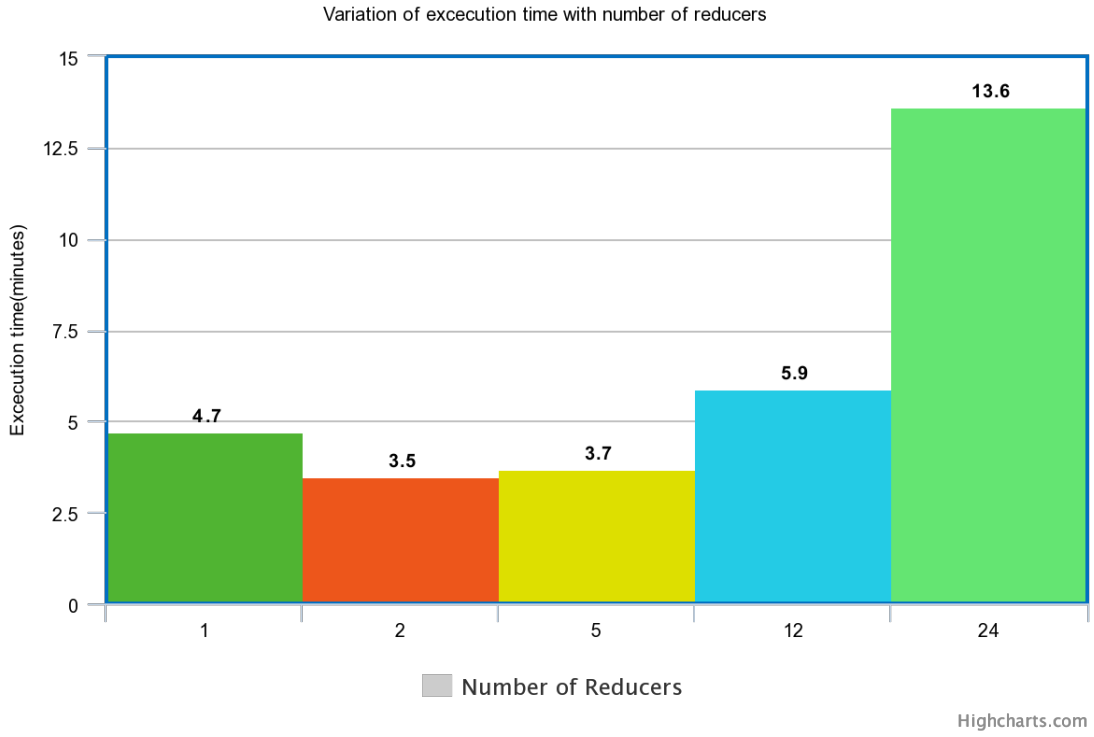
This is similar to the Most Visited Drops, only that we will add a check while generating key value pair whether the hour of time is [22,23,0,1,2]. Afterwards the same logic as most visited drops is followed.

2.1.6 Most Expensive Route

This is similar to the Most Popular Route, but instead of count of passengers, we will have the fare instead, which we will use in the place of count for the operations we did in Most Popular Route (i.e. checking if current route fare is greater than top 5, and updating the max cost by checking with the incoming key-value pair).

3. Variation of number of reducers

We know that the number of reducers is an important parameter to control in the map-reduce paradigm. We will examine the effect of changing values of number of reducers on the execution time. We chose the problem of most-popular route to analyze this trend. Also we reduce the data further by random sampling from 10 million to 5 million so that it is slightly easier to conduct the experiment. Here we assume that the trend of execution time with varying number of reducers is not linked directly to the size of the data.



The efficiency of reduces is driven by a large extent by the performance of the shuffle.The number of reduces configured for the application is, hence a crucial factor.Too few reduces cause undue load on the node on which the reduce is scheduled — in extreme cases, we have seen reduces processing over some GBs per-reduce. This also leads to very bad failure-recovery scenarios, since a single failed reduce, has a significant, adverse, impact on the latency of the job.

Too many reduces adversely affects the shuffle crossbar. Also, in extreme cases it results in too many small files created as the output of the job — this hurts both the NameNode and performance of subsequent Map-Reduce applications who need to process lots of small files.

4. Variation of slow start parameter

The reduce phase has 3 steps: shuffle, sort, reduce. Shuffle is where the data is collected by the reducer from each mapper. This can happen while mappers are generating data since it is only a data transfer. On the other hand, sort and reduce can only start once all the mappers are done. We can tell which one MapReduce is doing by looking at the reducer completion percentage: 0-33 percent means its doing shuffle, 34-66 percent is sort, 67-100 percent is reduce. This is why our reducers will sometimes seem "stuck" at 33 percent– it's waiting for mappers to finish.

Reducers start shuffling based on a threshold of percentage of mappers that have finished. We can change the parameter to get reducers to start sooner or later.

Why is starting the reducers early a good thing? Because it spreads out the data transfer from the mappers to the reducers over time, which is a good thing if your network is the bottleneck. The following graph shows the variation of execution time for the popular-route problem.