

Project 2

Using a Hadoop Cluster to Crunch Data From New York City Cab System

Rashmit

COMP47470 Big Data Programming,
School of Computer Science,
University College Dublin, Belfield, Dublin 4, Ireland
rashmit@ucdconnect.ie

Objective -

In this project we intend to analyze Yellow Cab Data of New York City Taxi and Limousine Commission dataset. This dataset is immense and hence we used Map Reduce concept to process it to make our infrastructure and the running task efficient. We are also using our platform as a service in Microsoft Azure and hence our machines are in cloud and will be allocated only when the machines are in Running state. Following are the steps followed in this project:

1. Creating test infrastructure in Microsoft Azure.
2. Configuring connection between Masters and Slaves.
3. Inserting Dataset into the clusters.
4. Creating and running Map reduce scripts as per the questions.





HDFS, the Hadoop Distributed File System, is a distributed file system designed to hold very large amounts of data (terabytes or even petabytes), and provide high-throughput access to this information. Files are stored in a redundant fashion across multiple machines to ensure their durability to failure and high availability to very parallel applications.

Microsoft Azure, formerly known as Windows Azure, is Microsoft's public cloud computing platform. It provides a range of cloud services, including those for compute, analytics, storage and networking

Test Bed Creation -

Step 1: The first step would be to sign up in Microsoft Azure and get the valid key credentials. The account will be recharged with 170 Euros and will be valid for 1 month.

Step 2: The next step is to create Virtual Machines for the configuration of Master and slaves. The Virtual Machine Specs are as follows:

NAME	STATUS	RESOURCE GROUP	LOCATION	SUBSCRIPTION
 master	Running	master	West Europe	Free Trial
 slave1	Stopped (deallocated)	master	West Europe	Free Trial
 slave2	Stopped (deallocated)	master	West Europe	Free Trial
 slave3	Stopped (deallocated)	master	West Europe	Free Trial

Name: A2 Basic

Cores: 2

Memory (RAM): 3.5GB

Storage Disk Type: HDD

Storage Disk Size: 250 GB

Hadoop Version: Hadoop-2.7.2

Data Replication: 3

Step 3: Down load the dataset of the Yellow Cab data, from New York City Taxi And Limousine Commission, available at http://www.nyc.gov/html/tlc/html/about/trip_record_data.shtml. The

```
-rw-rw-r-- 1 comp47470 comp47470 2.0G Apr 13 11:53 yellow_tripdata_2015-03.csv
-rw-rw-r-- 1 comp47470 comp47470 1.8G Apr 13 11:55 yellow_tripdata_2015-06.csv
-rw-rw-r-- 1 comp47470 comp47470 2.0G Apr 13 11:55 yellow_tripdata_2015-05.csv
-rw-rw-r-- 1 comp47470 comp47470 2.0G Apr 13 11:56 yellow_tripdata_2015-04.csv
```

The dataset consists of .csv files for every month of the year and the size of the file ranges from 1.5GB to 2GB per file.

The metadata of the files is described in the following link:

http://www.nyc.gov/html/tlc/downloads/pdf/data_dictionary_trip_records_yellow.pdf

Field Name	Description
VendorID	A code indicating the TPEP provider that provided the record. 1= Creative Mobile Technologies, LLC; 2= VeriFone Inc.
tpep_pickup_datetime	The date and time when the meter was engaged.
tpep_dropoff_datetime	The date and time when the meter was disengaged.
Passenger_count	The number of passengers in the vehicle. This is a driver-entered value.
Trip_distance	The elapsed trip distance in miles reported by the taximeter.
Pickup_longitude	Longitude where the meter was engaged.
Pickup_latitude	Latitude where the meter was engaged.
RateCodeID	The final rate code in effect at the end of the trip. 1= Standard rate 2=JFK 3=Newark 4=Nassau or Westchester 5=Negotiated fare 6=Group ride
Store_and_fwd_flag	This flag indicates whether the trip record was held in vehicle memory before sending to the vendor, aka "store and forward," because the vehicle did not have a connection to the server. Y= store and forward trip N= not a store and forward trip
Dropoff_longitude	Longitude where the meter was disengaged.
Dropoff_latitude	Latitude where the meter was disengaged.
Payment_type	A numeric code signifying how the passenger paid for the trip. 1= Credit card 2= Cash 3= No charge 4= Dispute 5= Unknown 6= Voided trip
Fare_amount	The time-and-distance fare calculated by the meter.
Extra	Miscellaneous extras and surcharges. Currently, this only includes the \$0.50 and \$1 rush hour and overnight charges.
MTA_tax	\$0.50 MTA tax that is automatically triggered based on the metered rate in use.
Improvement_surcharge	\$0.30 improvement surcharge assessed trips at the flag drop. The improvement surcharge began being levied in 2015.
Tip_amount	Tip amount – This field is automatically populated for credit card tips. Cash tips are not included.
Tolls_amount	Total amount of all tolls paid in trip.
Total_amount	The total amount charged to passengers. Does not include cash tips.

Configuring Multi Node (Masters and Slaves) -

Step 1: Configuring all the machines with single node configuration. Following steps were followed:

1. Install a Java environment (JVM and SDK)
2. Install SSH
3. Generate self key - In this case, we need to generate key only for master node. We will then use this key in other 3 slave nodes.
4. Prepare hadoop files
5. Edit .bashrc file (environment variables)
6. Update hadoop configuration files
7. Create folders for namenode and datanode
8. Format namenode

9. Start hadoop daemons and test

The End-points in Configuration files, Variables in .bashrc file, Java Installations should be carefully done to avoid Job Failures. Any of the configuration files that miss an end-point in them will result in Job Failure. We need to use the Master node key in the other slaves

```
$ cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
```

```
master@master:~$ start-dfs.sh
17/04/30 19:37:23 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Starting namenodes on [localhost]
localhost: starting namenode, logging to /usr/local/hadoop/logs/hadoop-master-namenode-master.out
localhost: starting datanode, logging to /usr/local/hadoop/logs/hadoop-master-datanode-master.out
Starting secondary namenodes [0.0.0.0]
0.0.0.0: starting secondarynamenode, logging to /usr/local/hadoop/logs/hadoop-master-secondarynamenode-master.out
17/04/30 19:38:16 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
master@master:~$ start-yarn.sh
starting yarn daemons
starting resourcemanager, logging to /usr/local/hadoop/logs/yarn-master-resourcemanager-master.out
localhost: starting nodemanager, logging to /usr/local/hadoop/logs/yarn-master-nodemanager-master.out
master@master:~$ mr-jobhistory-daemon.sh start historyserver
starting historyserver, logging to /usr/local/hadoop/logs/mapred-master-historyserver-master.out
```

Step 2: The Replication property in the hdfs-site.xml file describes how many copies of each chunk needs to be maintained over the Multi-node Cluster, minimum being 1.

Replication is necessary in cases of Slave machine failure. If all the Slaves containing a particular chunk of data are down, the Map-Reduce Job fails.

Each File is split into chunks of 128MB each and distributed over the Data nodes (Slaves). Cluster configuration is one of the most important pre-requisites for running any kind of Map-Reduce job.

The End-points in Configuration files, Variables in .bashrc file, Java Installations should be carefully done to avoid Job Failures. Any of the configuration files that miss an end-point in them will result in Job Failure.

Configuration also involves setting up Static VPN in Azure Portal, getting the SSH keys right for all the Virtual Machines.

Multi-node cluster configuration -

The following Steps were done for master and slaves:

- a) Edit the /etc/hosts file and /etc/hostname file
- b) Update hadoop configuration files \$HADOOP_HOME/etc/hadoop
- c) Edit core-site.xml file, replacing localhost as master
- d) Edit hdfs-site file, replacing value 1 as 3 (replication factor), and remove dfs.namenode.name.dir property section
- e) Edit yarn-site.xml file
- f) Create datanodes directories and give ownership
- g) Reboot master

The following steps were done only in the Master node:

- a) Edit the file masters and the file slaves, remove localhost
- b) Edit the hdfs-site.xml, replace dfs.datanode.data.dir property to dfs.namenode.name.dir, including the directory name.
- c) Create namenode and give ownership
- d) Format the namenode (the exit status should be 0)
- e) Test the network connection between the nodes, using ping, and test the ssh connection from the server to all the nodes.
- f) Start the hadoop deamons

Map-Reduce Scripts:

The Map- Reduce scripts are written in python and each job works using two scripts in this project, the mapper and a reducer.

For running Python Map-Reduce Scripts, additional Hadoop-Streaming Jar is required which needs to be passed with an argument.

```
/share/hadoop/mapreduce/hadoop-streaming-3.0.0-alpha2.jar -file mapper.py  
-mapper mapper.py -file reducer.py -reducer reducer.py -input /data/taxi/*  
-output PassengerStats
```

To analyze the Questions, 3 set of Map-Reduce Scripts are implemented. One set each for Number of Passengers, Trip Distance and Payment Type.

All the averages are calculated in each set. Mapper reads each line and extracts useful information from it which can then be collated by the Reducer to answer the questions.

The Mapper Output for PassengerStats Mapper is:

The first field here is the Day of the Week, 0 being Monday through 6 being Sunday.

```
4      0      00      2      1  
4      0      00      5      1  
4      0      00      1      1  
4      0      00      1      1
```

The second field is the Month Number, 0 being January through to 11 being December.

The third field is the Hour of travel, in 24-hour format.

The fourth field is the Number of Passengers per trip.

The last field is the count for the trip, to be summed in the Reducer.

The first field acts as the Key and the rest of the fields can be referred to as the Values associated with the key.

The Reducer takes this input, combines data for each day and accordingly computes averages.

```

Monthly Stats
January    1.6994

Through the Week Stats
Friday     1.6994

Weekdays Vs Weekends

      Days      Ends
00:00-01:00 - 1.8295 | 0.0
09:00-10:00 - 1.6285 | 0.0

Overall Average Passengers per Trip: 1.6994

```

The Reducer prints all averages per Day of the Week, per hour on Weekdays & Weekends, per Month as well as overall Average Passengers per trip.

The Map-Reduce Scripts for Number of Passengers Question and Trip Distance Question only differ in one parameter passed in by mapper. For Trip Distance, the mapper passes the Trip Distance per trip instead of the Number of Passengers in the case above.

The Remaining Script is for the Payment type. The Mapper and Reducer Outputs for Payment Types are straightforward. The Mapper passes the Payment Type ID, the Reducer converts it into Names.

2	1	
1	1	
1	1	
2	1	

Credit Card	16
Cash	32
No Charge	1

1= Credit card
 2= Cash
 3= No charge
 4= Dispute
 5= Unknown
 6= Voided trip

The Payment Legend is obtained from the NYC commissions website.

Since the data is split across data nodes, the program moves to the data instead of the other way round which is traditional processing.

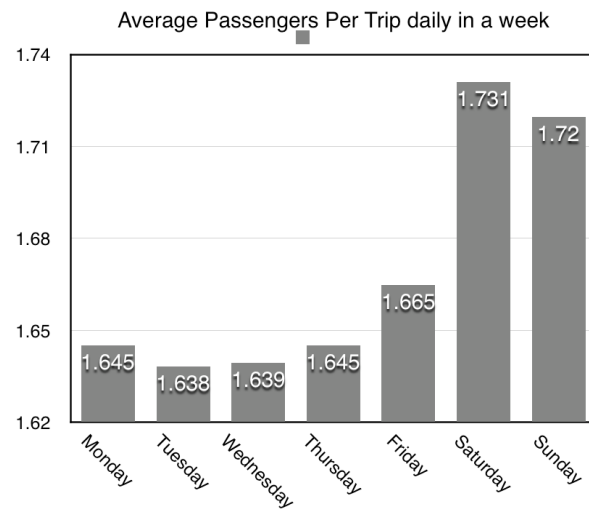
Multiple mappers are called on every Data Node and output is generated which becomes input for the Reducer. The reducer usually is called one time, which can also be configured.

Both the Mappers and Reducers work simultaneously based on the RAM and processing power of each Data Node. If a Map/Reducer crashes, it is executed again on the copy of the data chunk.

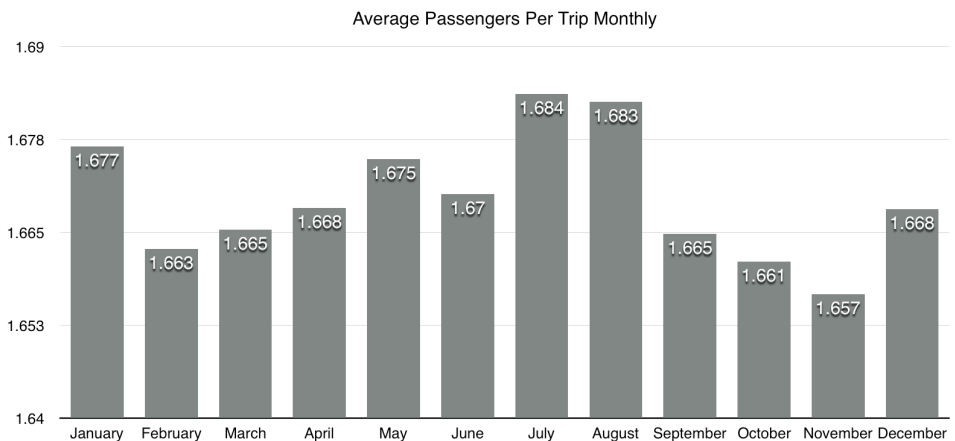
Answered Queries

1. What is the average number of passengers per trip in general, per month, and per day of the week?

Weekly Status	
Monday	1.6451
Tuesday	1.6383
Wednesday	1.6394
Thursday	1.6451
Friday	1.6649
Saturday	1.731
Sunday	1.7196

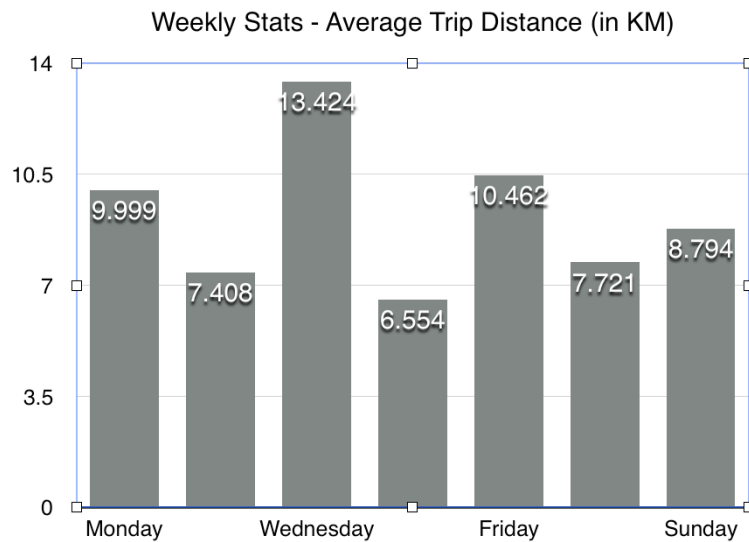


Average Passengers Per Trip monthly	
January	1.6766
February	1.6628
March	1.6654
April	1.6683
May	1.6749
June	1.6702
July	1.6836
August	1.6826
September	1.6648
October	1.6611
November	1.6567
December	1.6681

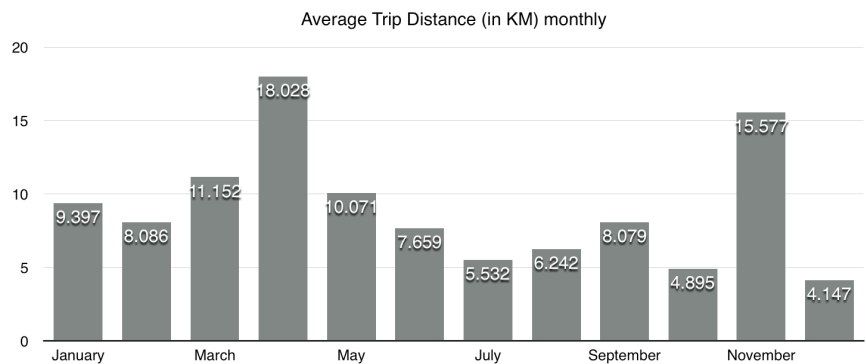


2. What is the average trip distance in general, per month, and per day of the week?

Average Trip Distance weekly	
Monday	9.9989
Tuesday	7.4077
Wednesday	13.424
Thursday	6.554
Friday	10.4621
Saturday	7.7211
Sunday	8.7939



Average Trip Distance monthly	
January	9.3967
February	8.0862
March	11.1519
April	18.0275
May	10.0707
June	7.6592
July	5.5317
August	6.2419
September	8.0794
October	4.8946
November	15.5771
December	4.1473

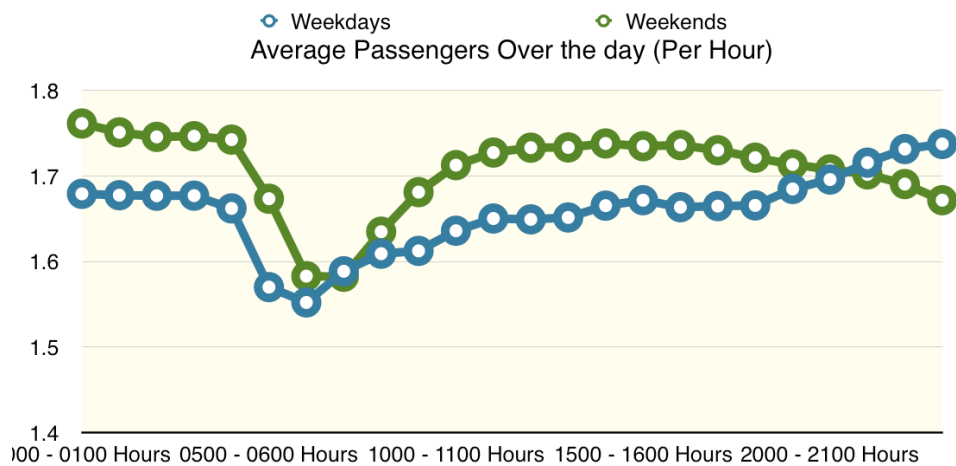


3. What is the most used payment type? Create an ordered list for payment types.

Most Common Payment Type (Ordered List)	
Credit Card	46,066,860
Cash	22,975,922
No Charge	270,402
Dispute	93,330
Unknown	6

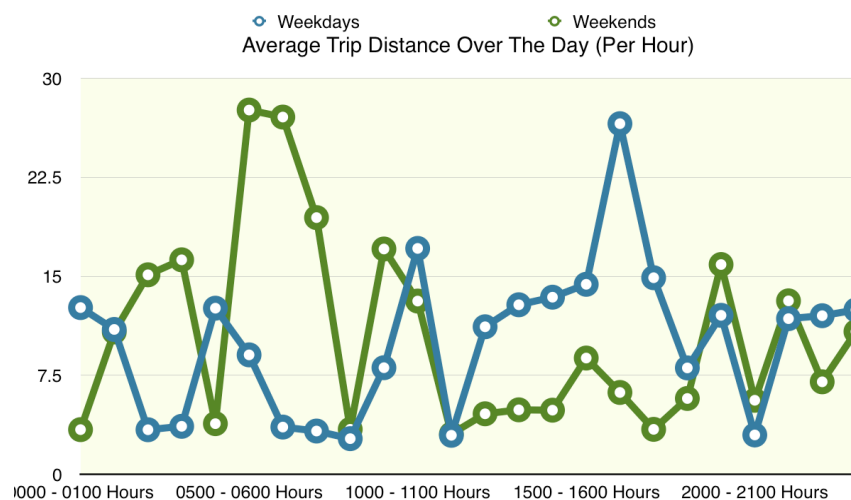
4. Create a graph showing the average number of passengers over the day (per hour). Create a version for workdays and another for weekend days.

Hours	Weekdays	Weekends
0000 - 0100 Hours	1.679	1.7611
0100 - 0200 Hours	1.6774	1.7508
0200 - 0300 Hours	1.6768	1.7455
0300 - 0400 Hours	1.6769	1.7463
0400 - 0500 Hours	1.662	1.7423
0500 - 0600 Hours	1.5701	1.6733
0600 - 0700 Hours	1.5522	1.5829
0700 - 0800 Hours	1.5886	1.5822
0800 - 0900 Hours	1.6091	1.6348
0900 - 1000 Hours	1.6124	1.681
1000 - 1100 Hours	1.6359	1.7125
1100 - 1200 Hours	1.6502	1.7272
1200 - 1300 Hours	1.6492	1.7329
1300 - 1400 Hours	1.6514	1.7332
1400 - 1500 Hours	1.6652	1.7378
1500 - 1600 Hours	1.6716	1.7346
1600 - 1700 Hours	1.6634	1.7359
1700 - 1800 Hours	1.6647	1.7299
1800 - 1900 Hours	1.6655	1.7212
1900 - 2000 Hours	1.6847	1.7133
2000 - 2100 Hours	1.6957	1.7075
2100 - 2200 Hours	1.7154	1.7017
2200 - 2300 Hours	1.7313	1.6904
2300 - 2400 Hours	1.7372	1.6716



5. Create a graph showing the average trip distance over the day (per hour). Create a version for workdays and another for weekend days.

Hours	Weekdays	Weekends
0000 - 0100 Hours	12.6278	3.3834
0100 - 0200 Hours	10.9727	10.7804
0200 - 0300 Hours	3.3752	15.1111
0300 - 0400 Hours	3.6347	16.2461
0400 - 0500 Hours	12.5827	3.8367
0500 - 0600 Hours	9.0465	27.5905
0600 - 0700 Hours	3.5775	27.0599
0700 - 0800 Hours	3.2776	19.4441
0800 - 0900 Hours	2.7005	3.4089
0900 - 1000 Hours	8.0747	17.0672
1000 - 1100 Hours	17.1098	13.1228
1100 - 1200 Hours	2.9546	3.0536
1200 - 1300 Hours	11.1748	4.5845
1300 - 1400 Hours	12.8414	4.8833
1400 - 1500 Hours	13.4143	4.8578
1500 - 1600 Hours	14.3918	8.8065
1600 - 1700 Hours	26.5574	6.1989
1700 - 1800 Hours	14.8899	3.4048
1800 - 1900 Hours	8.0592	5.744
1900 - 2000 Hours	12.0415	15.8894
2000 - 2100 Hours	2.981	5.6078
2100 - 2200 Hours	11.7908	13.1382
2200 - 2300 Hours	12.0126	7.0053
2300 - 2400 Hours	12.4499	10.8093



Challenges/Outcomes

- The first challenge was getting the valid key for the Azure account which created a challenge
- In the configuration, had missed some end-points in one of the Configuration files due to which the HDFS did not start. To resolve this I went through each of the steps and Config files again.
- Cloning the Virtual Machines seemed to be taking a very long time. This was understood as Azure had to compress the OS with all its settings and actually transfer it to 3 more machines. Their Stock Ubuntu deployment was much faster. Therefore, I created 4 single node Hadoop systems first without cloning, and then carefully configured them into a Multi-Node Cluster. Avoiding Cloning process saved more time for me.
- System Crashes on performing the Operation on more data (4-5 years) on the VMs was another challenge. To find the right balance between the type of Machine and the Volume of data to be processed was time consuming. Also, keeping in mind the Cost and Core Limitations.

Conclusion

- Working on this project did give a lot of experience and knowledge of working on Large Volumes of Real-World data using the Hadoop Framework.
- In addition, I also learned more about Linux Systems and Working on the Cloud.