Report:

Using a Hadoop Cluster to Crunch Data

From New York City Cab System

INTRODUCTION:

Processing big data sets once required supercomputers and other pricey, specialized hardware. Hadoop makes reliable, scalable, distributed computing possible on industry-standard servers–allowing you to tackle petabytes of data and beyond on smaller budgets. Hadoop is also designed to scale from a single server to thousands of machines, and detect and handle failures at the application layer for better reliability.

Microsoft Azure:

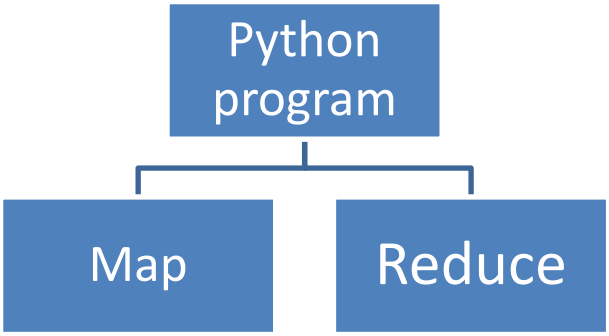
Microsoft Azure is a growing collection of integrated cloud services that developers and IT professionals use to build, deploy, and manage applications through our global network of datacenters. Azure CLI 2.0 is optimized for managing and administering Azure resources from the command line, and for building automation scripts that work against the Azure Resource Manager.

REQUIREMENT:

The requirement of the project is to form clusters using Microsoft azure platform. The cluster should have one name node and 3 data nodes with a replication factor of 3. A python program must be created to run a map reduce function on the data that is downloaded data. The data is Yellow Cab data, from New York City Taxi and Limousine Commission dataset.

DESCRIPTION:

The map reduce is done in python which is installed in the master node. There are two files.



Map.py:

Preprocessing the data:

We first remove the headers and blank space that is done at the beginning as we need to remove these in the .csv data files. Then we remove the white spaces that are there in the file. Then we separate each word as tokens that are separated by comma (,).

We take the vendor name separately(line 0) , trip distance(line 4),fare amount(line 12),payment(line 11),date(line 10),passenger count(line 3)

We then create dictionaries to store the mapped data of each vendor according to the trip distance, fare amount etc. The dictionaries are:

- Trip per vendor
  The trip distance will be appended to this dictionary at every iteration
- Fare amount vendor
  Fare amount of each vendor will be appended to this dictionary
- Payment methods
  We append the total number of times each payment method was used per vendor
- Month in year
  Month will be the key and the trip distance per month will be kept in this dictionary
- Passenger per vendor
  We keep the passenger count per vendor here

Output:

Creating clusters output:

```
AccountType    Location    Name       ProvisioningState    ResourceGroup    TimeCreated
-------------  ----------  ---------  -------------------  ---------------  -------------------------------
Standard_LRS   westeurope  diskimage  Succeeded            ratheeshS        2017-05-01T20:45:42.538080+00:00
```

```
0413164925.vhd
{
  "accountType": "Premium_LRS",
  "creationData": {
    "createOption": "Import",
    "imageReference": null,
    "sourceResourceId": null,
    "sourceUri": "https://ratheeshsdisks442.blob.core.windows.net/vhds/Hadoop20170413164925.vhd",
    "storageAccountId": null
  },
  "diskSizeGb": null,
  "encryptionSettings": null,
  "id": "/subscriptions/b6623394-1d83-4c9c-b7ea-d95aeaeaa736/resourceGroups/ratheeshS/providers/Microsoft.Compute/disks/diskimage",
  "location": "westeurope",
  "name": "diskimage",
  "osType": null,
  "ownerId": null,
  "provisioningState": "Succeeded",
  "resourceGroup": "ratheeshS",
  "tags": null,
  "timeCreated": "2017-05-01T20:45:42.538080+00:00",
  "type": "Microsoft.Compute/disks"
}
```

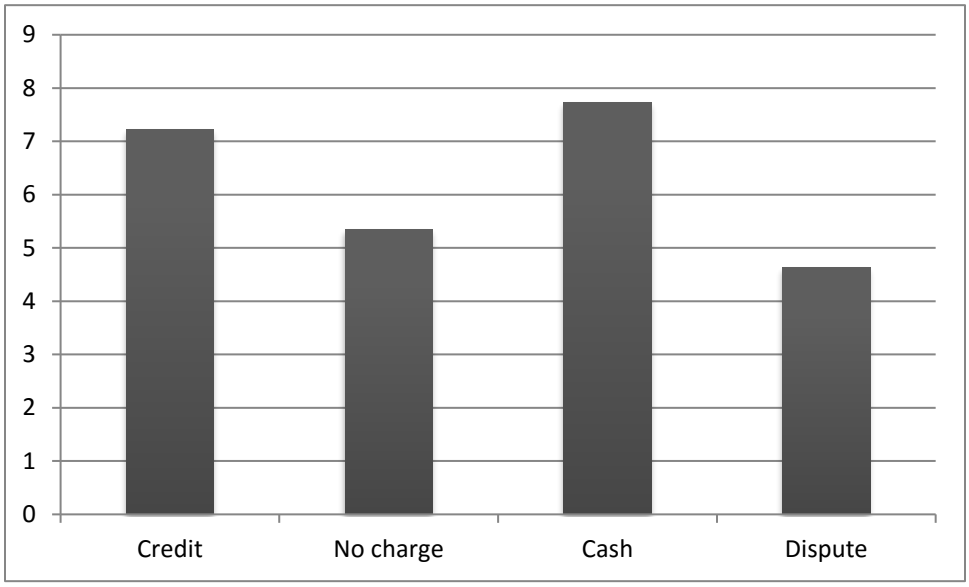
NAME	STATUS	RESOURCE GROUP	LOCATION	SUBSCRIPTION
Hadoop	Running	ratheeshS	West Europe	Free Trial
slave1	Running	ratheeshS	West Europe	Free Trial
slave2	Running	ratheeshS	West Europe	Free Trial
slave3	Running	ratheeshS	West Europe	Free Trial

Output of the program:

The picture below shows the pandas data frame that was made with both date and its values.

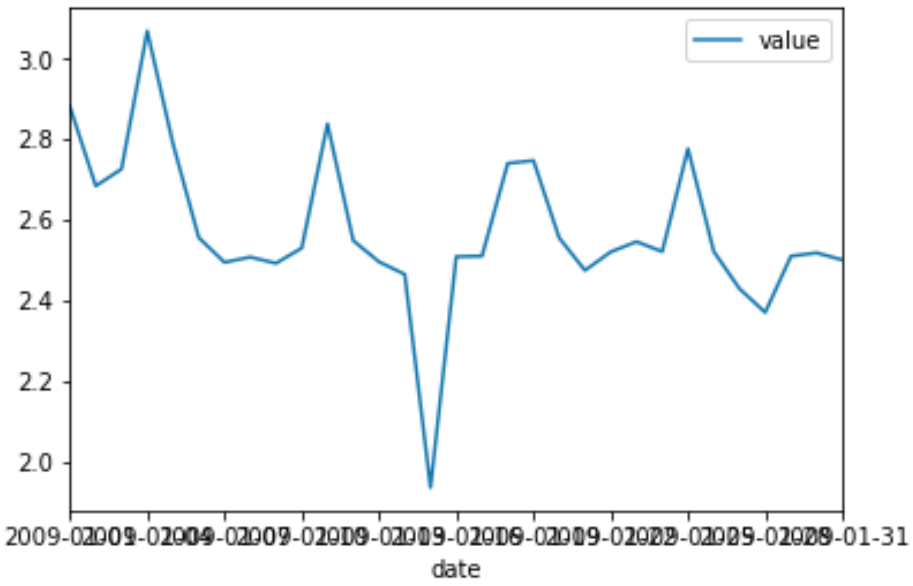
date	value
2009-01-01	2.880037
2009-01-02	2.683653
2009-01-03	2.725801
2009-01-04	3.068747
2009-01-05	2.788340
2009-01-06	2.555307
2009-01-07	2.493943
2009-01-08	2.506939
2009-01-09	2.491602
2009-01-10	2.529427
2009-01-11	2.838084
2009-01-12	2.547967
2009-01-13	2.495249
2009-01-14	2.464112
2009-01-15	1.934632
2009-01-16	2.508103
2009-01-17	2.509516
2009-01-18	2.739903
2009-01-19	2.746791
2009-01-20	2.554472
2009-01-21	2.473944
2009-01-22	2.519892
2009-01-23	2.545285
2009-01-24	2.520524
2009-01-25	2.776410
2009-01-26	2.521061
2009-01-27	2.428439
2009-01-28	2.369344
2009-01-29	2.509451
2009-01-30	2.517442
2009-01-31	2.499874

The chart below shows how the payments are distributed according to each payment method



The chart below shows how the average number of customers ride in each day of month in

```
<matplotlib.axes._subplots.AxesSubplot at 0x114f04ba8>
```



Challenges Faced:

1) Creating a VM

Tried creating a single node initially with Azure Portal. Two main difficulties were faced:

- Creating the VM in student discount. The options were not clear and an amount of 30 Euros was deducted from the bank account.

Description	Paid out
Friday, 7th April 17	
VDP-MSFT *AZURE	- 30.07

So choosing of the appropriate plan was execute this project was difficult.

- Many difficulties were faced when installing Azure CLI in Windows environment. Tried re-installing python and also supporting applications such as node.js, but it did not work. Hence after the VM was obtained "putty" was used to install the single node cluster. But a linux machine was used to create master and slaves and then connect them in a vnet.

2) Creating a cluster:

- The master node's port number had to be removed from the hdfs-site.xml as this led to an error when I tried to SSH the master node and name node did not start because it was in the master node.
- The disks were found that they could be downloaded to the local HDD if the "ActiveSAS" state was enabled. But this led to an error when the clusters were created and had to be unchecked.

3) Executing the program:

The data was too large to be downloaded hence only the 2011 data was taken. But even then the data was large so only 6 months data was taken. This was evident from the factor that mapper ran 100% but the reducer could not run and resulted in memory errors.

To solve this error the virtual memory was increased by increasing the RAM of the VM's but still it resulted in this error.

While executing the number of passengers and other count values the initial value had to be set as 1 and then the values were added together in the reducer phase, this resulted in a long duration. I prefer the use of combiner would have solved this duration problem as it would have improved the input to the reducer.

This error could not be solved hence I have installed Hadoop in my local system and downloaded the 2011 data and ran the map reduce program on it.

Conclusion:

Choosing the memory , both hard disk and virtual memory served to be a hard task , this could have been better with Amazon AWS rather than Azure CLI which is difficult to manage with different OS. The data was sometimes in numeric and others in string. Such types of data should first be cleaned by either R or SPSS and then stored using Hadoop.