|  |  |
| --- | --- |
| |  | | --- | | https://mail.google.com/mail/u/0/images/cleardot.gif | |

                    BIGDATA - PROJECT REPORT



Mr. Aditya A. Takale

Under the guidance of - Prof. Kasralikar

Project Outline

Music Industry has reached a new high in Business and it has gone beyond the usual stereotypes of CD and DVD’s. It has become much more freely available on the internet, thanks to ever enhancing connectivity and newer technologies.

As every day passes, new recommendation engines have emerged in the market and they use BigData heavily in their processing. My motive behind Analyzing this Million Songs Dataset is to simulate a recommendation engine using Design Patterns used in Hadoop Mapreduce.

Index

Literature Review ………………………………………………………………………………….3

The Dataset …………………………………………………………………………………………..3

Project Architecture……………………………………………………………………………….4

Project Implementation………………………………………………………………………… **5**

Design Patterns………………………………………………………………………………………12

Steps to run the code……………………………………………………………………………..16

Conclusion………………………………………………………………………………………………17

References………………………………………………………………………………………………17

1. Literature Review

Link - <http://www-personal.umich.edu/~yjli/content/projectreport.pdf>

The Link to the above paper envisages a study to build a recommendation engine based on Million Songs Dataset. The Accuracy of recommendations is constantly monitored using a variety of Algorithms and visualizations techniques to reach to a single conclusion. The research has shown how it is possible to use song Metadata as well as the data from Users histories to provide the best set of songs to recommend.

Also, many different kinds of analysis were done including visualization for User-based approach, comparison of recommendation methods, User based Vs Listened song count, Item –based Vs Listened song count. Precision Matrix was drawn, KNN analysis was done as also Matrix Factorization was carried out. K means Clustering was done as well as analysis of User based recommendations was done based on the formula.

So, to conclude, it was said that the song-with-metadata approach yields low precision compared to the user-based and song-without-meta methods.

1. The Dataset

The Million Song Dataset is a freely-available collection of audio features and metadata for a million contemporary popular music tracks.

Its purposes are:

To encourage research on algorithms that scale to commercial sizes

To provide a reference dataset for evaluating research

As a shortcut alternative to creating a large dataset with APIs (e.g. The Echo Nest's)

To help new researchers get started in the MIR field

The core of the dataset is the feature analysis and metadata for one million songs, provided by The Echo Nest. The dataset does not include any audio, only the derived features.

The Million Song Dataset is also a cluster of complementary datasets contributed by the community:

SecondHandSongs dataset -> cover songs

musiXmatch dataset -> lyrics

Last.fm dataset -> song-level tags and similarity

Taste Profile subset -> user data

thisismyjam-to-MSD mapping -> more user data

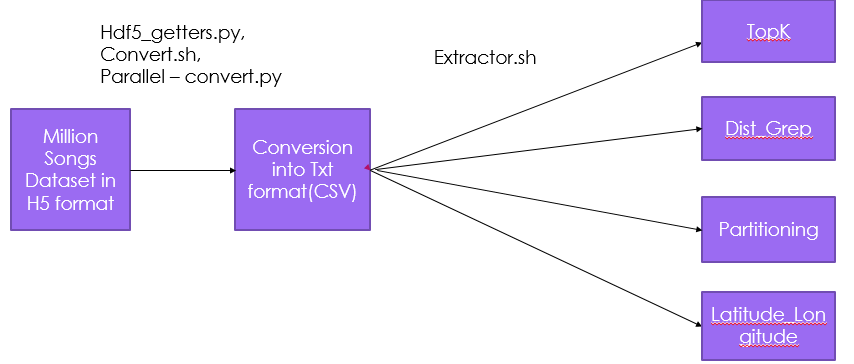
tagtraum genre annotations -> genre labels

Top MAGD dataset -> more genre labels

The Million Song Dataset started as a collaborative project between The Echo Nest and LabROSA. It was supported in part by the NSF.

The dataset contains about million records with 54 fields in each record in H5 format. This H5 format, I have converted into txt format with comma separated values. This format is processed further for the Bigdata Analysis. Size of Raw data about 1.3 GB. Contains more than 500,000 records. Has 54 fields including artist name, song name, Artist Id, Song Id etc. Original Data in H5 format.

The dataset also contains some helper files to better know the dataset.

1. Project Architecture

Above is the Architecture for the project. Here, the original Dataset available in h5 format is processed using Hdf5 \_getters.py, convert.sh and parallel-convert.py scripts to convert it into txt format(CSV). Once, we have the data in Txt format, this data is further processed using Extractor script to come up with Design Patterns such as TopK design Pattern. Then Dist\_Grep Design pattern, Partitioning Design Pattern and Latitude\_Longitude Design pattern. So, here the extractor script basically gives us only the desired fields and this makes the processing and filtering the data easy.

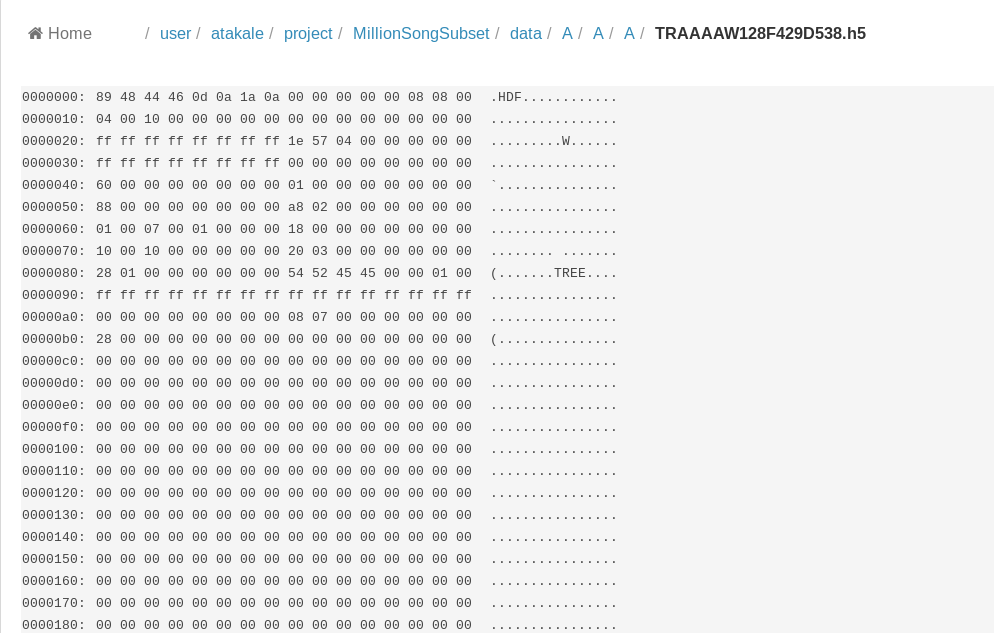
This is the basic idea to perform a very reliable and easy filtering on the data. Personally, I feel that this greatly reduces efforts and is quite efficient in terms of performance.

1. Project Implementation

Getting the Dataset:- <http://labrosa.ee.columbia.edu/millionsong/>

There were 2 approaches which I followed to get and process the dataset. In the first approach, I downloaded the dataset from above website. The size of the dataset was around GB’s. It was in HDF5 format.

The dataset in HDF5 format is absolute gibberish if unprocessed and is stored like a folder within a folder format. There were 2 main folder A and B, each containing folders A through Z in turn again contain folders A through Z . Likewise the data was arranged in a folder structure. Below is the Screen shot of how the data looked.



HDF5 format:-

Hierarchical Data Format (HDF) is a set of file formats (HDF4, HDF5) designed to store and organize large amounts of data.

HDF5 is a data model, library, and file format for storing and managing data. It supports an unlimited variety of datatypes, and is designed for flexible and efficient I/O and for high volume and complex data.

The HDF5 Technology suite includes tools and applications for managing, manipulating, viewing, and analyzing data in the HDF5 format.

It was Originally developed at the National Center for Supercomputing Applications.

This data which was in HDF5 format was required to be converted to Text for processing. Hence, I used a python script to convert it into txt file. Of the lines of code are.

import hdf5\_getters

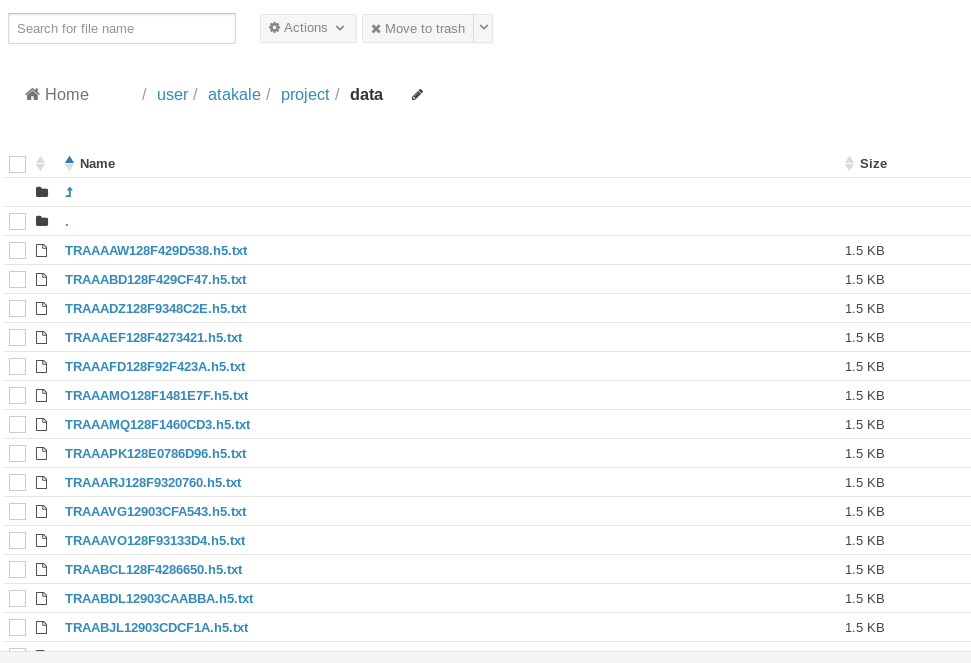
h5 = hdf5\_getters.open\_h5\_file\_read(path to some file)

duration = hdf5\_getters.get\_duration(h5)

h5.close()

This a small code snippet to extract the duration of song, Likewise all the fields can be obtained using the hdf5\_getters.py script. I have kept the hdf5\_getters file in the cluster.

Now, there was another problem. A single record came as a single file using this script.



In the above diagram, we can see that each file has size of 1.5 Kb and each consists of only 1 record. Likewise several records are presenting several folders. The Task was to assimilate data of all the files in a single file.

Hence, I wrote a shell script to process this data convert.sh.

========================================================================================

#!/bin/bash

sudo apt-get -fy install python-tables h5utils hdf5-tools &

wget -c http://static.echonest.com/millionsongsubset\_full.tar.gz

wget -cqb https://raw.githubusercontent.com/tbertinmahieux/MSongsDB/master/PythonSrc/hdf5\_getters.py

tar -xvzf millionsongsubset\_full.tar.gz

list=`find ./MillionSongSubset/data -iname \*.h5`

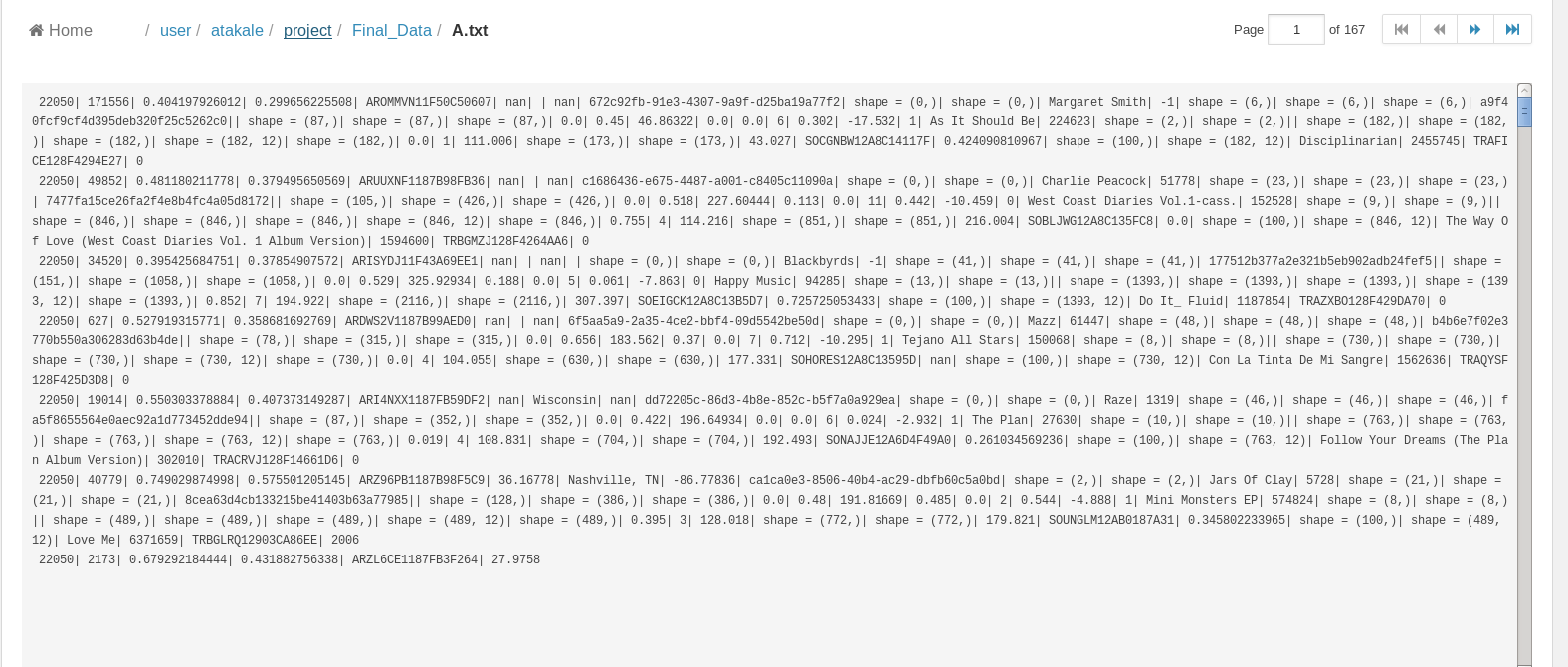
for line in $list

do

python display\_song.py $line > "$line.txt"

done

Because of this script, all the data was populated in one single file. Now, the data was readable and in Text format separated by “|”. Below is the screen shot of the same.



But, here the real problem was that the data size here was just 66 Mb.

Hence, a bigger Dataset was needed.

There was another full dataset which had a size of GB .Now an interesting thing about it was that there was an amazon instance on which the whole dataset was publicly available. The dataset is available as an Amazon Public Dataset snapshot which can easily be attached to an Amazon EC2 virtual machine to run experiments in the cloud. The instance could be simply set up as an EBS disk instance from snap-5178cf30

For me, when I launch an EC2 virtual machine running Ubuntu, then create an EBS instance from that snapshot, then attach the EBS to the virtual machine, it appears as /dev/xvdf from within Ubuntu. Then you just have to mount it.

Initially, I was processing using a single core instance which is available freely on the AWS. But then , the processing speed was very less. Hence, I bought a very powerful machine have 16 core processor with 64 Ghz processor. The processing was done very fast. I ran the hdf5\_getters.py script on the instance itself. Also, I ran the converter script to dump all the data from all the files into a single file. This was done using multi-threaded program. 26 threads were running in parallel to produce faster results.

===============================================================================

#!/bin/bash

count=0

for level0 in A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

do

count=$(( $count + 1 ))

for level1 in A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

do

for level2 in A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

do

list=`find /mnt/snap/data/$level0/$level1/$level2/ -iname \*.h5`

for line in $list

do

name=`echo $line | cut -d"/" -f8`

echo "$count: $name"

python hdf5\_getters.py $line > /mnt/mydata/my/$name.txt

done

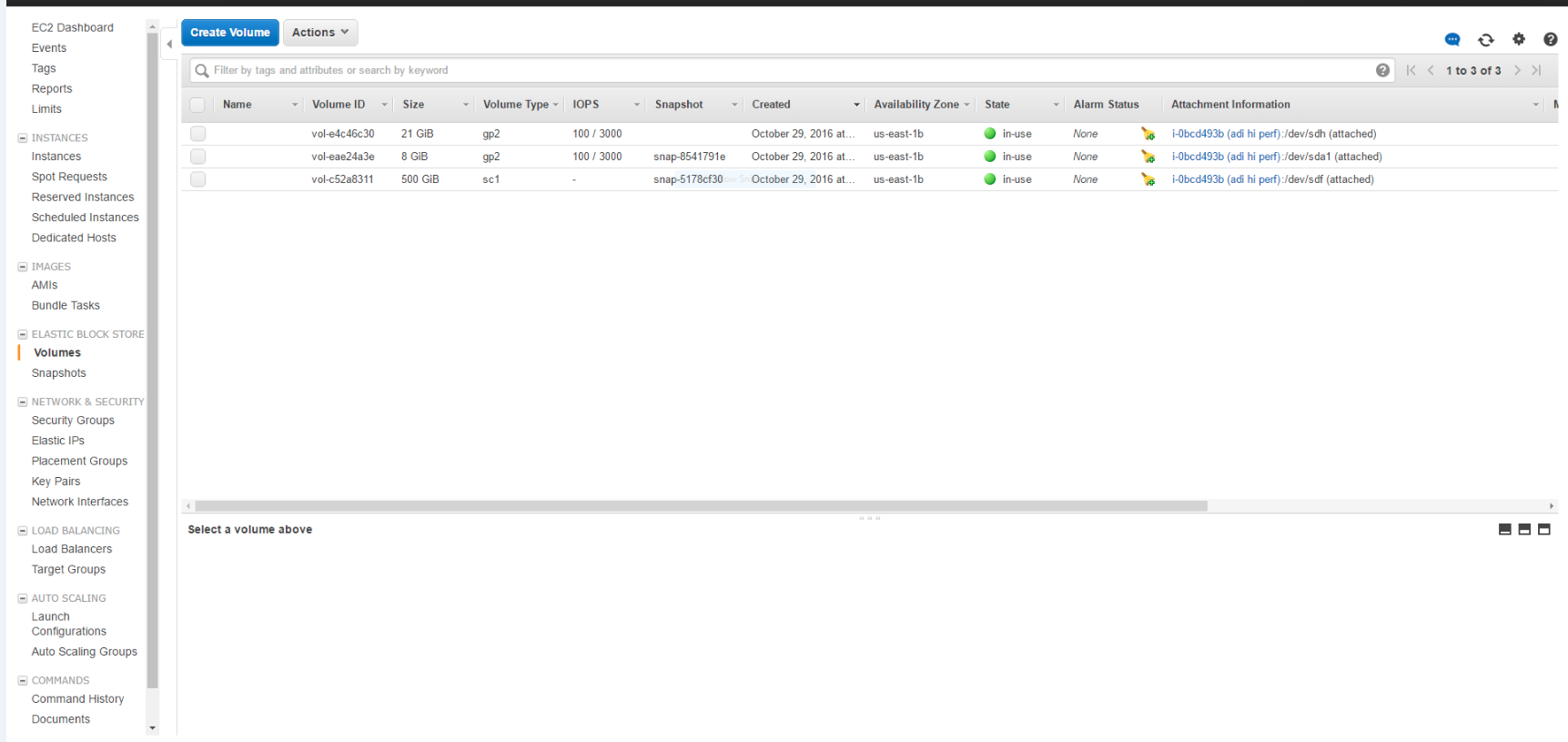
done

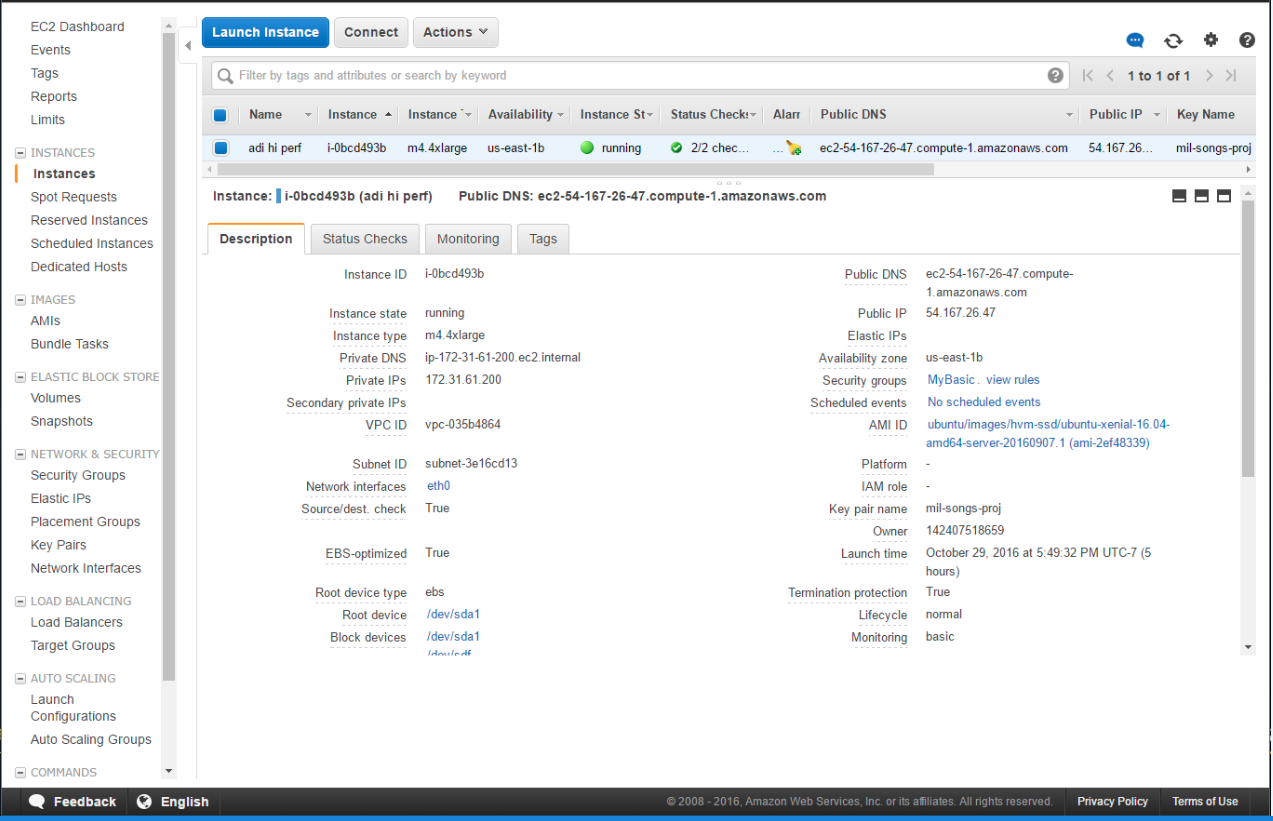
done &

done

=============================================================================

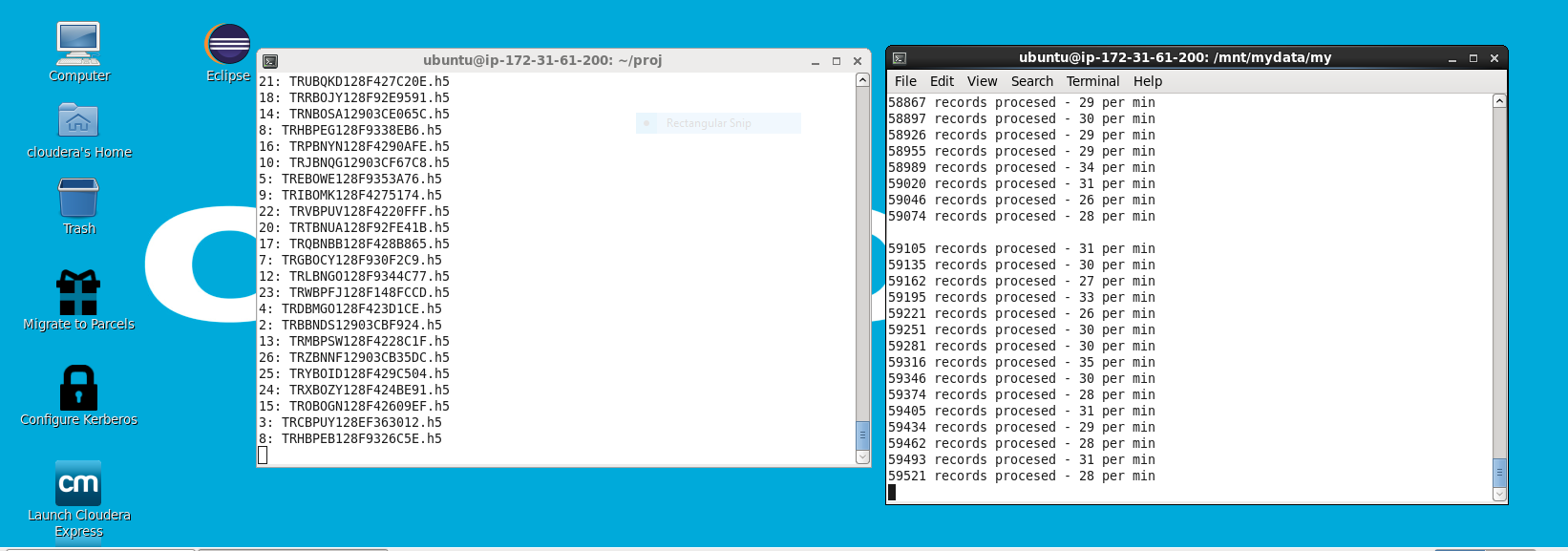
Also, some of the screen shots for AWS instance are shows as below.





Here, we can see the Instance configuration. Instance is 4x large. So, the processing was really fast.

Below is the screen shot of records getting processed in parallel simultaneously.



The left window shows which thread is running and the right one shows how many recordsare processed per second.

Likewise I had a raw dataset of around 1.3 Gb. After removing the Unicode characters and null values and gibberish data, I had a clean dataset of around 942 Mb.

This data was to be processed. I finally had a proper dataset.

On this dataset, I performed 4 kinds of analysis using the design patters.

The design patterns were.

1. TopK Design pattern
2. Distributed \_Grep
3. Summarization pattern(Latitude-Longitude)
4. Partitioning Pattern

It is important to remember that these design patterns were selected keeping in mind that I was trying to emulate a recommendation engine.

So, from a users point of view, selecting on of the Top 10 songs, searching based on Artist name, partitioning the Artists based on starting letter would be really helpful. Also, the summarization pattern for Latitude and Longitude was chosen for Visualization.

Also as mentioned earlier, extractor.sh was used to retrieve required fields from the dataset.

The code snippet is shown as below.

================================================================================================

list=`find ./data/ -iname \*.h5.txt`

echo "insert headersss here" > ~/Downloads/Aditya\_data\_all.csv

for row in $list

do

data1=`grep -i "analysis\_sample\_rate:" $row | cut -d":" -f2`

data2=`grep -i "artist\_7digitalid:" $row | cut -d":" -f2`

data3=`grep -i "artist\_familiarity:" $row | cut -d":" -f2`

data4=`grep -i "artist\_hotttnesss:" $row | cut -d":" -f2`

data5=`grep -i "artist\_id:" $row | cut -d":" -f2`

data6=`grep -i "artist\_latitude:" $row | cut -d":" -f2`

data7=`grep -i "artist\_location:" $row | cut -d":" -f2`

data8=`grep -i "artist\_longitude:" $row | cut -d":" -f2`

data9=`grep -i "artist\_mbid:" $row | cut -d":" -f2`

data10=`grep -i "artist\_mbtags:" $row | cut -d":" -f2`

data11=`grep -i "artist\_mbtags\_count:" $row | cut -d":" -f2`

data12=`grep -i "artist\_name:" $row | cut -d":" -f2`

data13=`grep -i "artist\_playmeid:" $row | cut -d":" -f2`

data14=`grep -i "artist\_terms:" $row | cut -d":" -f2`

data15=`grep -i "artist\_terms\_freq:" $row | cut -d":" -f2`

data16=`grep -i "artist\_terms\_weight:" $row | cut -d":" -f2`

data17=`grep -i "audio\_md5:" $row | cut -d":" -f2`

data18=`grep -i "bars\_confidencesong\_id:" $row | cut -d":" -f2`

data19=`grep -i "bars\_start:" $row | cut -d":" -f2`

data20=`grep -i "beats\_confidence:" $row | cut -d":" -f2`

data21=`grep -i "beats\_start:" $row | cut -d":" -f2`

data22=`grep -i "danceability:" $row | cut -d":" -f2`

data23=`grep -i "mode\_confidence:" $row | cut -d":" -f2`

data24=`grep -i "duration:" $row | cut -d":" -f2`

data25=`grep -i "end\_of\_fade\_in:" $row | cut -d":" -f2`

data26=`grep -i "energy:" $row | cut -d":" -f2`

data27=`grep -i "key:" $row | cut -d":" -f2`

data28=`grep -i "key\_confidence:" $row | cut -d":" -f2`

data29=`grep -i "loudness:" $row | cut -d":" -f2`

data30=`grep -i "mode:" $row | cut -d":" -f2`

data31=`grep -i "release:" $row | cut -d":" -f2`

data32=`grep -i "release\_7digitalid:" $row | cut -d":" -f2`

data33=`grep -i "sections\_start:" $row | cut -d":" -f2`

data34=`grep -i "sections\_confidence:" $row | cut -d":" -f2`

data35=`grep -i "segments\_confidencesong\_id:" $row | cut -d":" -f2`

data36=`grep -i "segments\_loudness\_max:" $row | cut -d":" -f2`

data37=`grep -i "segments\_loudness\_max\_time:" $row | cut -d":" -f2`

data38=`grep -i "segments\_loudness\_start:" $row | cut -d":" -f2`

data39=`grep -i "segments\_pitches:" $row | cut -d":" -f2`

data40=`grep -i "segments\_start:" $row | cut -d":" -f2`

data41=`grep -i "time\_signature\_confidence:" $row | cut -d":" -f2`

data42=`grep -i "time\_signature:" $row | cut -d":" -f2`

data43=`grep -i "tempo:" $row | cut -d":" -f2`

data44=`grep -i "tatums\_start:" $row | cut -d":" -f2`

data45=`grep -i "tatums\_confidence:" $row | cut -d":" -f2`

data46=`grep -i "start\_of\_fade\_out:" $row | cut -d":" -f2`

data47=`grep -i "song\_id:" $row | cut -d":" -f2`

data48=`grep -i "song\_hotttnesss:" $row | cut -d":" -f2`

data49=`grep -i "similar\_artists:" $row | cut -d":" -f2`

data50=`grep -i "segments\_timbre:" $row | cut -d":" -f2`

data51=`grep -i "title:" $row | cut -d":" -f2`

data52=`grep -i "track\_7digitalid:" $row | cut -d":" -f2`

data53=`grep -i "track\_id:" $row | cut -d":" -f2`

data54=`grep -i "year:" $row | cut -d":" -f2`

#echo "$data48|$data47|$data51|$data53|$data24|$data14|$data12|$data9"

#echo "$data9|$data12|$data14|$data24|$data47|$data49|$data51|$data53" >> ~/Downloads/Aditya\_data\_selective.csv

echo "$data1|$data2|$data3|$data4|$data5|$data6|$data7|$data8|$data9|$data10|$data11|$data12|$data13|$data14|$data15|$data16|$data17|$data18|$data19|$data20|$data21|$data22|$data23|$data24|$data25|$data26|$data27|$data28|$data29|$data30|$data31|$data32|$data33|$data34|$data35|$data36|$data37|$data38|$data39|$data40|$data41|$data42|$data43|$data44|$data45|$data46|$data47|$data48|$data49|$data50|$data51|$data52|$data53|$data54">> ~/Downloads/Aditya\_data\_all.csv

done

===============================================================================

This is the extractor script, in which I can extract the selected fields y commenting the others. The result will be “|” separated fields of data.

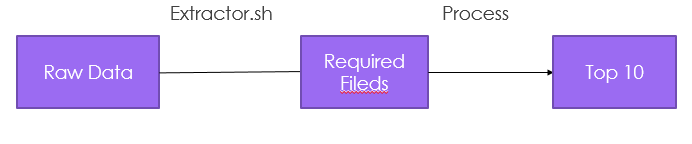
1. Design Patterns:-
2. Top 10 Songs

Each song in the dataset has a degree of hotness associated with it.

This degree of hotness is the measure of how popular the song was when it was released. This is associated with song\_id of the song.

Used the extractor script to extract these desired fields from the raw data.

This analysis was helpful in knowing the most popular songs in given dataset.



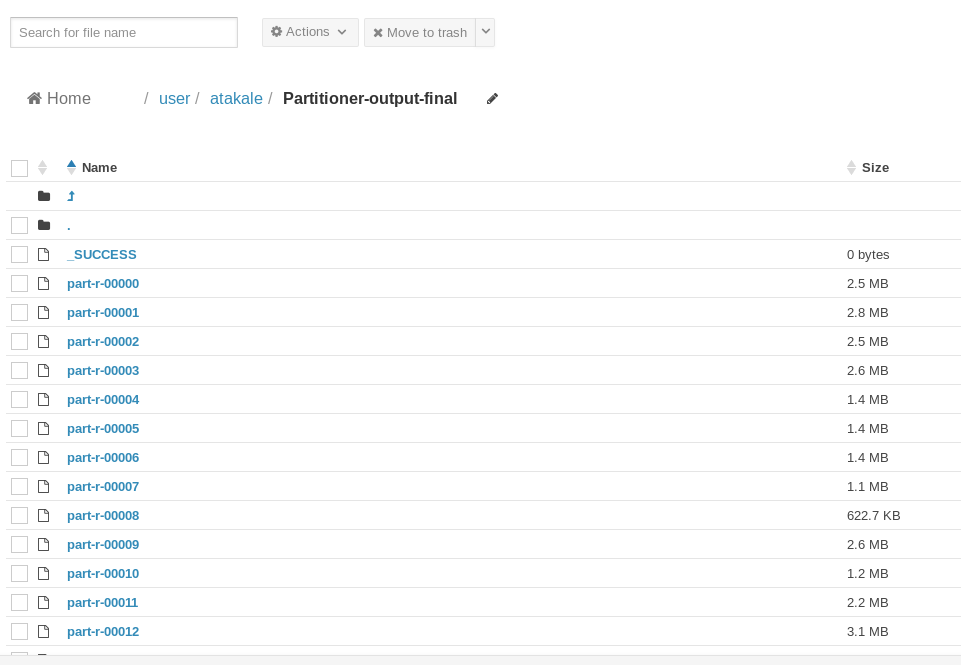
Result:

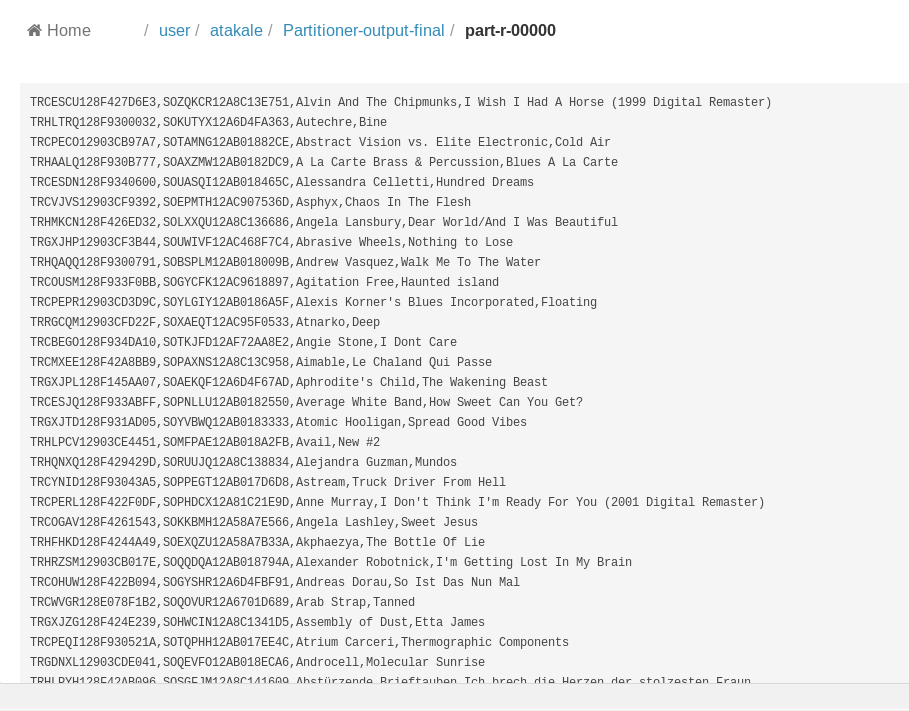


1. Partitioning Pattern

Each record has an artist name associated with it. based on Artist Name. Partitions were formed based on first letter of “Artist name”. Artist Id, Song Id, Artist Name and Song Name. Extractor was again used to fetch these 4 columns from the main dataset. This analysis was useful in grouping the Artists based on their names, so that finding their related data becomes easy.

Result:





1. Distributed\_Grep

Each record has an artist name associated with it.

Distributed\_Grep allows us to search a record based on Artist Name provided by the User.

Artist Id, Artist Name and Track\_id is populated.

Extractor was again used to fetch these 3 columns from the main dataset.

This analysis was useful to retrieve a information about artist, if the artist name is known.

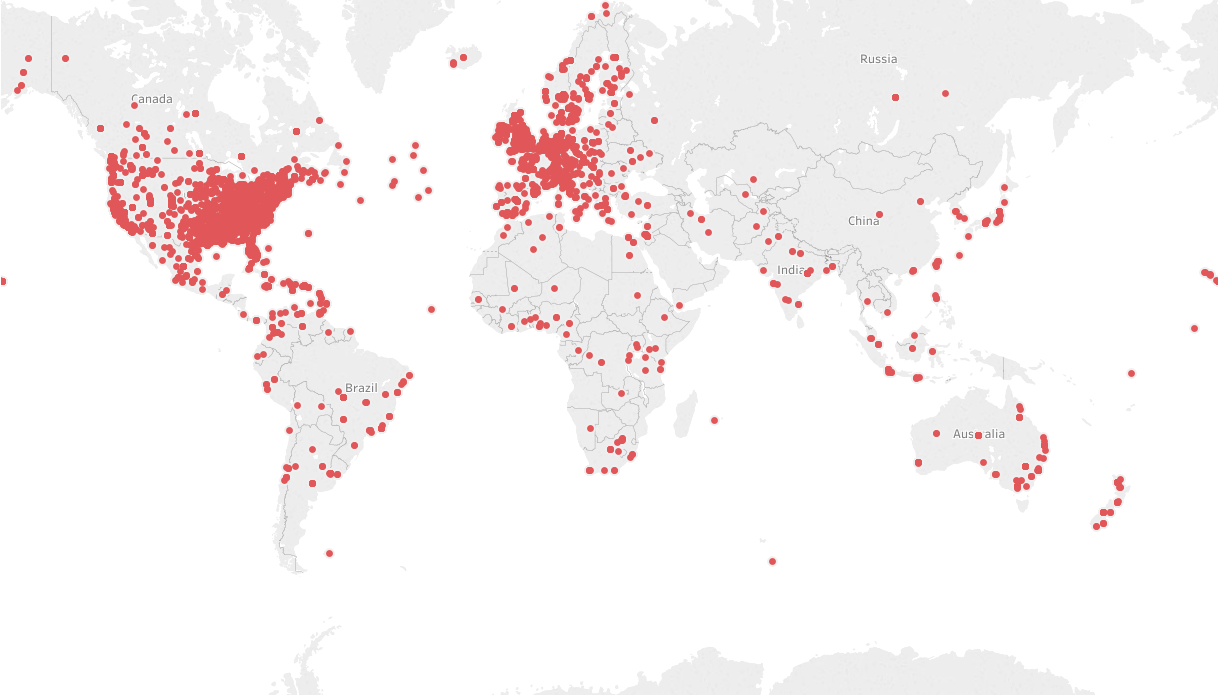
Result:-



1. Distributed \_Grep Pattern(Visualization using Tableau)

Each record has a latitude and longitude associated with the artist. Mapping was done for artists based on Latitude and Longitude and Key, value pairs were formed. This was collected in Microsoft excel and was used as an input for Tableau. Visualization shows red dots for latitudes and longitudes, when hovered over, it gives an artist corresponding artist name.

The Visualizations show that most of the artists in Million Songs Dataset are based in America, that too east coast. So, the or listeners are based in the east coast. Also, many artists are based in Europe. So, we can definitely say that most of the songs are either American or European.



The original Tableau file is stored at this address on Hadoop cluster:-

/ user/ atakale/ project/ Tableau/ Book1.twbx

6. Steps to run the code

1. logon to Vmware

2. Logon to the Gateway Machine using the SSH command. Then Enter User\_id and password.

Location of main data on hadoop cluster

/user/atakale/project/Million-Songs-Dataset-Final

location of sub datasets-

1./user/atakale/project/Data-for-TopTen

2./user/atakale/project/Data\_For\_Partitioner

3./user/atakale/project/Data\_For\_Dist\_Grep

4./user/atakale/project/Data-for-Latitude-Longitude

3. The paths where the corresponding jar files are stored-

1./home/atakale/Dist\_Grep/bigdata-0.0.1.jar

2./home/atakale/lat-long-final/bigdata-0.0.1.jar

3./home/atakale/partitioner-final/bigdata-0.0.1.jar

4./home/atakale/TopTen/bigdata-0.0.1.jar

The commands to run the code on Hadoop are as follows

1.hadoop jar /home/atakale/TopTen/bigdata-0.0.1.jar Aditya.Bigdata.TopTenDriver /user/atakale/project/Data-for-TopTen Top-Ten-Outpu-final

2.hadoop jar /home/atakale/ lat-long-final /bigdata-0.0.1.jar Aditya.Bigdata.TopTenDriver /user/atakale/project/ Data-for-Latitude-Longitude Latitude-Longitude-final

3.hadoop jar /home/atakale/Dist\_Grep/bigdata-0.0.1.jar Aditya.Bigdata.TopTenDriver /user/atakale/project/Data\_For\_Dist\_Grep Dist\_Grep-output-final

4.hadoop jar /home/atakale/partitioner-final/bigdata-0.0.1.jar Aditya.Bigdata.TopTenDriver /user/atakale/project/Data\_For\_Partitioner Partitioner-output-final

1. Conclusion:-

To conclude, I would say that Hadoop reduced the execution time significantly the processing time. While running the job on my local VM required more time as compared to running the job on Hadoop Cluster.

Also, based on Visualization, I could deduce that most of the artists were concentrated in east and west of America. And rating of these artists was quite high. The Top 10 songs only had artists present in the east coast.

1. References
2. <https://support.hdfgroup.org/HDF5/>
3. https://en.wikipedia.org/wiki/Hierarchical\_Data\_Format