Movie Recommendation System Using Hadoop and Spark

1. DESCRIPTION

The rapid rise of the World Wide Web and online entertainment delivery systems have led to a massive influx of data from various sources. While arts and entertainment were confined to offline access and consumption for the better part of the 20th century, over the past few decades it has evolved into something much more scalable. Fuelled by the growth in technology, movie sites such as Netflix have been at the forefront of content delivery systems.

Hadoop, Spark and other such big data processing frameworks have been helpful in crunching numbers for data intensive operations and deliver results as and when required. One such application which has been implemented through this project is the Movie Recommendation system.

A movie recommendation system scans the data accumulated over a period of years and recommends an unseen movie to a user who has similar tastes to users who have already seen the movie in question.

In our project, we have implemented a Movie Recommendation System using both Hadoop and Spark and tried to make a comparison between the two. We actually ran our code on 'big' data. We built a recommendation list by analyzing 10 Million ratings using Hadoop. We also analyzed 1 Million movie ratings on both Hadoop and Spark for the sake of comparison. The project uses Hadoop to run the MapReduce jobs on a set of virtual clusters on Future Systems, based on OpenStack. Python programming language was used to perform the map and reduce functions on a MovieLens dataset, taken from grouplens.org.

2. PROBLEM STATEMENT

We had the idea for our project from an assignment from Prof. Predrag's Data Mining class, where it was concluded that high performance computing resources are needed to build a recommendation system for datasets as large as the MovieLens 10M dataset. We decided to try this out and were successfully able to build a recommendation system by analyzing the MovieLens 10M dataset using Hadoop, using collaborative filtering and data mining algorithms.

3. PURPOSE AND OBJECTIVES

Following were our main objectives while working on this project:

1. To identify and group users with similar tastes and scan their preferences.
2. For a particular movie, check it the user in question has seen the movie. If not, find the rating of that particular movie with respect to users of similar tastes.
3. Based on similarity scores, suggest appropriate movies to users.
4. Using Hadoop, run the job parallely on multiple clusters to speed up the running time.
5. To analyze and collect meaningful results from the analysis of significantly large data.

6. To test the performance of Hadoop vs. Spark, which is a comparatively newer technology and is said to be up to 100 times more efficient than Hadoop.

4. APPROACH

Our criteria for model evaluation was as follows:

- For each user *i* and each movie *j* they have not seen, we find *k* most similar users who have seen *j* and then use them to infer user *i*'s rating on movie *j*.
- If we cannot find *k* users for some movie *j*, then we take all users who have seen it.
- We test the performance on our system using cross validation.

The "cosine similarity" coefficient has been used to identify the similarity value rating, ranging from 0 (meaning no similarity) to 1 (extremely high similarity).

5. RESULTS

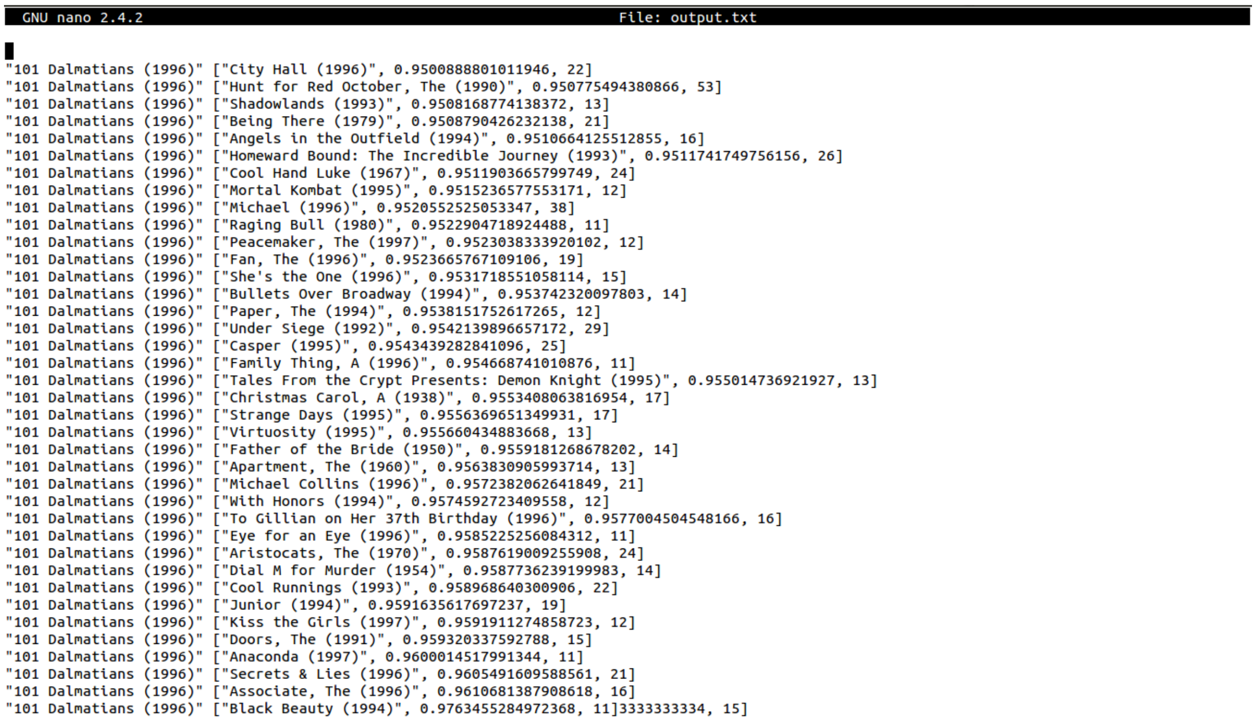
For our project's implementation on Hadoop, we used the 10 Million Ratings dataset. We ran our code using 10 *m1.small* virtual instances on the Amazon AWS service. Our results were as follows:

- We were able to generate a 1.2 Gigabytes results file – 'simslarge.txt'
- For every movie in the dataset, this file contained a list of movies that were most recommended for that particular movie.
- With 10 virtual instances, it took us around 16 hours to run our code.
- We also ran the same code on the 1 Million dataset using 4 virtual instances, and it took us around 60 minutes to do that.

For the Spark part, we ran our code on the 1 Million Ratings dataset, using 5 *m3.xlarge* virtual instances on the Amazon AWS service. We designed the code such that on each run, we had to enter a particular movie as input, and the output yielded the most recommended movies for our input movie. To load our files onto Amazon AWS, we used Amazon S3 bucket. We created a new bucket, loaded our ratings data, movies data and the source code onto it, and gave the reference to our bucket while running the final code. Our results for the Spark implementation were as follows:

- It took around 20 minutes to generate results for a particular movie.
- We tested the code for different movies, and on all our trials it took between 15 and 20 minutes.

Here, we attach a snapshot of what our results file looks like:

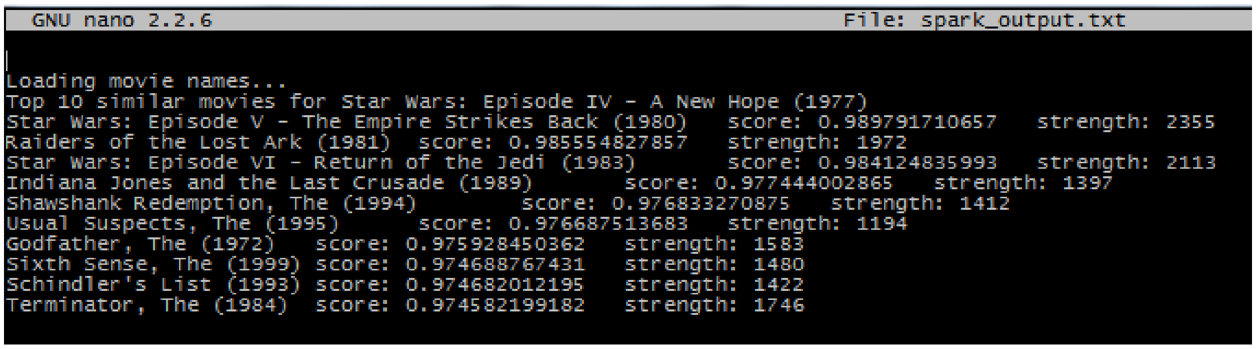


```
GNU nano 2.4.2                                   File: output.txt

"101 Dalmatians (1996)" ["City Hall (1996)", 0.9500888801011946, 22]
"101 Dalmatians (1996)" ["Hunt for Red October, The (1990)", 0.950775494380866, 53]
"101 Dalmatians (1996)" ["Shadowlands (1993)", 0.9508168774138372, 13]
"101 Dalmatians (1996)" ["Being There (1979)", 0.9508790426232138, 21]
"101 Dalmatians (1996)" ["Angels in the Outfield (1994)", 0.9510664125512855, 16]
"101 Dalmatians (1996)" ["Homeward Bound: The Incredible Journey (1993)", 0.9511741749756156, 26]
"101 Dalmatians (1996)" ["Cool Hand Luke (1967)", 0.9511903665799749, 24]
"101 Dalmatians (1996)" ["Mortal Kombat (1995)", 0.9515236577553171, 12]
"101 Dalmatians (1996)" ["Michael (1996)", 0.9520552525053347, 38]
"101 Dalmatians (1996)" ["Raging Bull (1980)", 0.9522904718924488, 11]
"101 Dalmatians (1996)" ["Peacemaker, The (1997)", 0.9523038333920102, 12]
"101 Dalmatians (1996)" ["Fan, The (1996)", 0.9523665767109106, 19]
"101 Dalmatians (1996)" ["She's the One (1996)", 0.9531718551058114, 15]
"101 Dalmatians (1996)" ["Bullets Over Broadway (1994)", 0.953742320097803, 14]
"101 Dalmatians (1996)" ["Paper, The (1994)", 0.9538151752617265, 12]
"101 Dalmatians (1996)" ["Under Siege (1992)", 0.9542139896657172, 29]
"101 Dalmatians (1996)" ["Casper (1995)", 0.9543439282841096, 25]
"101 Dalmatians (1996)" ["Family Thing, A (1996)", 0.954668741010876, 11]
"101 Dalmatians (1996)" ["Tales From the Crypt Presents: Demon Knight (1995)", 0.955014736921927, 13]
"101 Dalmatians (1996)" ["Christmas Carol, A (1938)", 0.9553408063816954, 17]
"101 Dalmatians (1996)" ["Strange Days (1995)", 0.9556369651349931, 17]
"101 Dalmatians (1996)" ["Virtuosity (1995)", 0.955660434883668, 13]
"101 Dalmatians (1996)" ["Father of the Bride (1950)", 0.9559181268678202, 14]
"101 Dalmatians (1996)" ["Apartment, The (1960)", 0.9563830905993714, 13]
"101 Dalmatians (1996)" ["Michael Collins (1996)", 0.9572382062641849, 21]
"101 Dalmatians (1996)" ["With Honors (1994)", 0.9574592723409558, 12]
"101 Dalmatians (1996)" ["To Gillian on Her 37th Birthday (1996)", 0.9577004504548166, 16]
"101 Dalmatians (1996)" ["Eye for an Eye (1996)", 0.9585225256084312, 11]
"101 Dalmatians (1996)" ["Aristocats, The (1970)", 0.9587619009255908, 24]
"101 Dalmatians (1996)" ["Dial M for Murder (1954)", 0.9587736239199983, 14]
"101 Dalmatians (1996)" ["Cool Runnings (1993)", 0.958968640300906, 22]
"101 Dalmatians (1996)" ["Junior (1994)", 0.9591635617697237, 19]
"101 Dalmatians (1996)" ["Kiss the Girls (1997)", 0.9591911274858723, 12]
"101 Dalmatians (1996)" ["Doors, The (1991)", 0.959320337592788, 15]
"101 Dalmatians (1996)" ["Anaconda (1997)", 0.9600014517991344, 11]
"101 Dalmatians (1996)" ["Secrets & Lies (1996)", 0.9605491609588561, 21]
"101 Dalmatians (1996)" ["Associate, The (1996)", 0.9610681387908618, 16]
"101 Dalmatians (1996)" ["Black Beauty (1994)", 0.9763455284972368, 11]3333333334, 15]
```

The above snapshot shows the movies that are most likely to be recommended for the movie *101 Dalmatians (1996)*. We can see that the movie *City Hall (1996)* has been most highly recommended for *101 Dalmatians (1996)* with a similarity score of 0.95 and 22 co-raters. Since this snapshot is only a sample analysis, we used the 100k dataset. Hence the number of co-raters are so few. In the actual 1M or 10M dataset, the number of co-raters will be significantly higher.

The following snapshot shows the results for the Spark code:



```
  GNU nano 2.2.6                              File: spark_output.txt

Loading movie names...
Top 10 similar movies for Star Wars: Episode IV - A New Hope (1977)
Star Wars: Episode V - The Empire Strikes Back (1980)   score: 0.989791710657   strength: 2355
Raiders of the Lost Ark (1981)  score: 0.985554827857   strength: 1972
Star Wars: Episode VI - Return of the Jedi (1983)       score: 0.984124835993   strength: 2113
Indiana Jones and the Last Crusade (1989)       score: 0.977444002865   strength: 1397
Shawshank Redemption, The (1994)        score: 0.976833270875   strength: 1412
Usual Suspects, The (1995)      score: 0.976687513683   strength: 1194
Godfather, The (1972)   score: 0.975928450362   strength: 1583
Sixth Sense, The (1999) score: 0.974688767431   strength: 1480
Schindler's List (1993) score: 0.974682012195   strength: 1422
Terminator, The (1984)  score: 0.974582199182   strength: 1746
```

In the above snapshot we can see that the most highly recommended movie for *Star Wars: Episode IV – A New Hope (1977)* is *Star Wars: Episode IV – The Empire Strikes Back (1980)*, with a similarity score of 0.9897 and 2355 co-raters. This snapshot was taken by analyzing the 1M dataset.

6. FINDINGS

- In our results, we tested our output file for many movies to confirm the accuracy of our results. For example, we tested our results against the movie *Star Wars: A New Hope (1977)*. According to our results, the most recommended movies for it were *Star Wars: The Empire Strikes Back (1980)* and *Star Wars: Return of the Jedi (1983),* and *Indiana Jones and the Last Crusade (1989).* Most of the movies recommended for *Star Wars: A New Hope (1977)* belonged to either the sci-fi or the action genre. We also tested our results against many popular movies, and for each movie, we mostly got recommendations of movies from the same genre.

- Since we implemented our project on both Hadoop and Spark, and ran it on sufficiently 'big' data, we are in a position to be able to compare the efficiencies of Hadoop and Spark. We discovered that Spark is a much more efficient and faster method of analyzing big data. In our project, we were able to analyze the 1 Million dataset in as less as 15 minutes, when using only 5 instances, while in Hadoop, while using a similar number of instances, it took us around an hour to do this.

  Of course, it has to be kept in mind that Hadoop can be run on small sized instances, while for Spark we are required to have at least *m3.xlarge* sized instances, i.e. Spark consumes a lot more memory than Hadoop does, which is one of the reasons of its efficiency.

  Thus, if the data is extremely large and the amount of resources available is significantly large, then Spark should be preferred over Hadoop. However, for datasets as large as a million, Hadoop can be used, albeit at the expense of time. According to our findings, using Hadoop beyond that is impractical.

7. INSTRUCTIONS

1. Make sure the ssh-agent is up and running. Also ensure that pip and git are installed. This instruction manual is with respect to the Chameleon cloud.

2. Clone the project repository to the local working directory.

3. Run the setup.sh file provided in the project repository. This will clone the big-data-stack and install all the necessary requirements

```
command::: $bash setup.sh
```

4. Now move into the big-data-stack folder that has been cloned and edit the .cluster.py file as follows:

```
a.  Change the openstack/image : CC-Ubuntu14.04
b.  Change the openstack/create_floating_ip: True
c.  Change the openstack/flavor: m1.medium
d.  Scroll down and change N_MASTER=8 and N_DATA=8
```

e. Set the following configuration:

```
_zookeepernodes = [(master, [0,1,2,3,4,5,6,7])]
_namenodes = [(master, [0,1,3,4])]
_journalnodes = [(master, [0,1,2,3,4,5,6,7])]
_historyservers = [(master, [2,3])]
_resourcemanagers = [(master, [0,1,3,4])]
_datanodes = [(master, xrange(N_DATA))]
_frontends = [(master, [0])]
_monitor = [(master, [2,3,4])]
```

5. Now run the boot.sh file which will boot up 8 m1.medium instances on the Chameleon cloud.
   ```
   command::: $bash boot.sh
   ```

6. Now, make sure a virtual environment has been activated and test if all nodes can be pinged.
   ```
   command::: $ansible all –m ping
   ```

7. If all the 'pongs' are received, from the project directory run the site.yml ansible script using the following command:
   ```
   command::: $ansible-playbook -i inventory.txt site.yml
   ```

   The above command will install pip and mrjob on all the nodes and also copy the python files needed for hadoop and spark execution.

8. Now, run the hadoop.yml file from the project directory which will run the python file on the remote name node and fetch the results back to the control node.
   ```
   command::: $ansible-playbook -i inventory.txt hadoop.yml
   ```

9. Use:
   ```
   command::: $nano  results/master0/home/hadoop/new_hadoop_output.txt
   ```

   to view the results of running the movie recommendation system python code on the 10 million ratings dataset.

10. Run the spark-hdfs.yml ansible script to copy the necessary data into hdfs for spark execution to proceed.
    ```
    command::: $ansible-playbook -i inventory.txt spark-hdfs.yml
    ```

11. Now, log into the frontendnode as per the inventory.txt file and run the following commands:
    ```
    command::: $sudo su – hadoop
    command::: $spark-submit --master yarn --deploy-mode client --executor-memory 3g
    Movie-Similarities-1m.py 260 > spark_output.txt
    ```

12. Now check the 'spark_output.txt' file for the recommendations for the given movie 260.

Notes:

1. If the hadoop.yml file does not run as expected, kindly set the $JAVA_HOME path in /opt/hadoop/etc/hadoop/hadoop-env.sh to /usr/lib/jvm/java-7-openjdk-amd64 manually. Also, due to configuration issues of ansible with spark, the python file has to be run by logging into the front end node for spark execution.

2. There are no problems with hadoop execution and it will run through ansible itself and fetch the results back to the control node as well. The hadoop execution results can be found at: results/master0/home/hadoop

3. A directory 'test' has been included in the 'src' directory containing the scripts to run the project on smaller datasets for faster results.

In case any of the ansible playbooks do not run for some reason, please follow these manual commands (these commands have to be executed after completing commands 1 to 6 listed above):

1. To install hadoop on all the nodes
   ```
   command::: $ansible-playbook -i inventory.txt play-hadoop.yml
   ```
2. To install spark
   ```
   command::: $ansible-playbook -i inventory.txt play-alladdons.yml
   ```
3. To transfer the python files to the frontendnode for execution
   ```
   command::: $scp MovieSimilaritiesLarge.py hadoop@<front-end-node-
   ip>:MovieSimialaritiesLarge.py

   command::: $scp Movie-Similarities-1m.py hadoop@<front-end-node-ip>:Movie-
   Similarities-1m.py
   ```
4. On all the nodes as root
   ```
   command::: $sudo su –
   command::: $sudo apt-get install python-pip
   ```
5. On all the nodes as root
   ```
   command::: $pip install mrjob
   ```
6. On all nodes as root set $JAVA_HOME in /opt/hadoop/etc/hadoop/hadoop-env.sh to /usr/lib/jvm/java-7-openjdk-amd64
7. Download the 10 million ratings dataset and unzip it
   ```
   command::: $sudo su – hadoop
   command::: $wget http://files.grouplens.org/datasets/movielens/ml-10m.zip
   command::: $unzip ml-10m.zip
   ```
8. Run the following command to run the python for hadoop execution:
   ```
   command::: $python MovieSimilaritiesLarge.py -r hadoop --items=ml-
   10M100K/movies.dat ml-10M100K /ratings.dat --no-conf --hadoop-bin
   /opt/hadoop/bin/hadoop > hadoop_output.txt
   ```
9. Copy the data into hdfs for spark execution:
   ```
   command::: $/opt/hadoop/bin/hdfs dfs -put ml-10M100K/ratings.dat
   hdfs://futuresystems/ratings.dat
   ```

10. Run the following command for spark execution:

```
command::: $spark-submit --master yarn --deploy-mode client --executor-memory 3g
Movie-Similarities-1m.py 260 > spark_output.txt
```

11. Check the hadoop output in the hadoop_output.txt file and spark output in the spark_output.txt file.

8. CONCLUSION

We successfully managed to carry out our idea – build a recommendation system for movies by analyzing practically large datasets. Our focus was on analyzing datasets as big as 10 Million, which we successfully did. As shown in our results, we also managed to achieve a satisfactory level of accuracy, since our recommender system recommends movies quite similar to the movie being tested. We would like to add our interpretation of Big Data technologies to our concluding statement. As already discussed, we found that Spark, compared to Hadoop, is much more efficient, although it also needs more resources. Hence, if the resources available to the user are sufficiently large, and the data is quite big, then we recommend using Spark over Hadoop for efficient results.

9. REFERENCES:

1. Python Code help for Mapreduce and Spark: Taming Big data with Hadoop and Spark, Frank Kane, Udemy.
2. Dr. Predrag's assignment from Data Mining Class