Recommender System using Collaborative Filtering and Performance Analysis

Divya Ginjupalli, Gopikrishnan Narayanan, Prabal Kanodia, Priyanka, SivaSrinivas Amara
Department of Computer and Information Science
University of Florida
Gainesville, USA

ABSTRACT

Recommender systems are widely used commercially and academically that involve predicting user response to certain options. Since the recommendation algorithms are computational intensive and usually operate on large datasets, such problems can be efficiently dealt with cloud architecture. The algorithms could be broadly classified as Content-based and Collaborative-based. Content-based systems recommend items based on the properties of the items whereas Collaborative filtering systems recommend items based on similarity measures between users and/or items. This paper aims at building a recommendation engine using Amazon AWS technologies (Hadoop, HDFS, Map-Reduce) with focus on collaborative filtering (CF) algorithms and performs comparative studies using Mahout library. Emphasis is made on predicting user-movie rating and recommending movies to the users based on user ratings using Netflix and IMDB dataset.

Keywords— recommender; collaborative filtering; CF; map-reduce; machine learning; mahout; dataset; merging; item-based; user-based; visualization; data analysis; D3.js

I. INTRODUCTION AND MOTIVATION

With rapid increase of information in today's world of Internet, it has become harder to make a choice or a decision. Recommender Systems are subclass of information filtering systems that predicts the rating or preferences that can be given to an item that is not yet considered by the user. This prediction is done using models based on the users past rating or preferences or item characteristics.

There are different approaches to make a recommendation system. Some of them are Item hierarchy based recommendation, Content based (attributes, properties) recommendation, Collaborative filtering item-item similarity, Collaborative filtering user-user similarity, Social interest graph based, Model based recommendation (Matrix factorization techniques). Item hierarchy based filtering recommends items based on the same hierarchy level. For example a person who has bought printer might be recommended ink for it. Content-based systems recommend items based on the similar properties of all other items. For example, if a Netflix user watched action movies starring *Clint EastWood* then the user would be suggested *Good Bad and Ugly*. Collaborative filtering recommends items based on user and/or item similarity using k-means, Pearson correlation coefficient[3], Cosine similarity or Jaccard Index. User similarity measure identifies users similar to the active user and recommends items they have rated the same together. Item similarity[2] measure identifies pairs of items and infers the taste of the active user. In social and graph based sites like Facebook, the system might recommend you *LadyGaga* because your friends have liked it. Model based (latent semantic model) approach unearths the latent features of the dataset and uses these features to recommend items to the user. The Hybrid based system uses both collaborative filtering and content- based filtering techniques may be together or just combining the results.

We aim to implement a recommender system using the Item-based and User-based collaborative filtering algorithms. We used a supervised learning algorithm for training and testing the data set. Apart from this we used existing Mahout Libraries to address the same problem on the same data set. Finally a comparison was done between these two versions of implementations to check the performance.

II. RELATED WORK

Scalding is an open source MapReduce framework by twitter. This abstraction layer over MapReduce eases the developer's task to implement parallel programs. It is written in Scala and provides higher-level functions like group, join, and filter to avoid rewriting patterns and help calculate similarities.

We emphasize on movie recommendation engine. There are existing engines like Netflix, MovieLens IMDB, and Rotten Tomatoes. Netflix and MovieLens ask users to rate movies to determine which movies you want to view next. IMDB automatically recommends similar movies you searched for. Rotten Tomatoes recommends movies based on a search criteria (preferred actors, directors, genres). There are other movie engines like Jinni that recommend movies over an array of search criteria (mood, plot places, audience, style genres) while Taste Kid recommends items in other categories like books, music and games along with similar movies.

We aim to visualize the recommendations and compare the performance of our algorithms against Mahout's using extended Netflix dataset.

There are other recommendation engines that recommend music (Last.fm), news (YCombinator), products (Amazon) etc and use one or more filtering and ranking algorithms. Other

technologies that have been applied to recommender systems includes Bayesian networks, clustering and Horting (graph-based technique in which nodes are users and edges represent the degree of similarity between two users)

III. FRAMEWORKS AND TECHNOLOGIES USED

In this paper we are developing a recommender system and implementing it through map-reduce using Amazon's cloud services on Hadoop framework for recommending and comparing these recommendations with the recommendations given using Mahout Library.

A. *Cloud computing*

Scalability, one of the big problems of CF is the motivation to use cloud architecture. As the volume of the data set is large the computational cost is also high. Cloud computing which is suitable for such a problem has been chosen to overcome the problem of large scale computation task. Cloud computing is a kind of service that provides dynamically scalable resources. Often these are provided as virtual resources over the internet. The resources include delivery of software, infrastructure and storage. We used IaaS(infrastructure-as-a-service) version of cloud computing.

B. *Map Reduce*

It is a programming paradigm for parallel processing of large data sets in distributed environment using basically two steps: Map and Reduce. Map function processes a key/value pair to generate a set of intermediate key/value pairs, and a reduce function merges all intermediate values associated with the same intermediate key. This leads to Automatic parallelism while executing on a large cluster of commodity machines [5].

C. *Hadoop*

It is a framework that allows distributed processing of large data sets on clusters of computers which provide local storage and computation using simple programming models like Map Reduce.

D. *HDFS*

The Hadoop Distributed File system is used to produce high throughput access to application data. Data in a Hadoop cluster is broken down into smaller pieces (called blocks) and distributed throughout the cluster. Hence Map Reduce functions can be performed on these subsets which provide scalability for large data sets.

E. *Mahout*

It is a library that provides implementations for distributed or otherwise called scalable machine learning and data mining algorithms.

F. *D3.js*

It is a JavaScript library to visualize and manipulate data driven documents. It allows us to bind the data to a Document Object Model (DOM), and then apply data-driven transformations to the document [20].

IV. SYSTEM ARCHITECHTURE

Collaborative filtering is the technique of filtering the information based on the collaboration among multiple agents, data points. Collaborative filtering uses the assumption that people with similar tastes will rate things similarly and users will give rating to the catalogue implicitly or explicitly. The two approaches user-user similarity and item-item similarity that fall under this category will be implemented.

A. *User-user similarity*

This is based on calculating the similarity between the users. The steps to follow in this approach:

- Find top 'k' users who are similar to the active user using Cosine/Pearson similarity
- Similarity measure is calculated over the ratings given to each common item between the user
- The corresponding (k-users)-item matrices are aggregated to predict a rating, the active user may give to a movie
- Suggest movies to the active user based on top ratings predicted

B. *Item-item similarity*

This approach recommends the item based on the "nearest" item to the items the user has rated. The steps for this approach are:

- Construct an item-item matrix to determining their co-occurrences based on user's history
- Infer the taste of the active user by examining the matrix and items consumed by the active user
- Suggest other items that occur more frequently with the items already consumed by the active user

C. *Matrix Factorization - Singular value decomposition[17]*

This approach recommends by characterizing items and users with a more holistic goal to uncover the latent features.

- Identify relevant latent features
- Compute dot product of user latent feature vector and item latent feature vector.
- Unearths latent features (ex: genre of a movie) of user-item
- Construct an rating matrix where value in each cell is the rating given by $user_i$ to $item_j$
- Factorize the matrix in reduced space (r) to find feature vectors (user preference feature vector and item preference feature vector) to predict a rating for an active user

$$D_{m*n} = U_{m*r}\, S_{r*r}\, V^{t}_{r*r}$$

In *Figure 1, DataSet Processor* component processes a large set of Netflix movie files. Each file lists the movieid and its associated ratings by different users. The processor formats and whitens the dataset using MapReduce. In addition to text formatting of ratings files, movie titles file is extended with genres. This processed dataset is input to the recommendation engine.

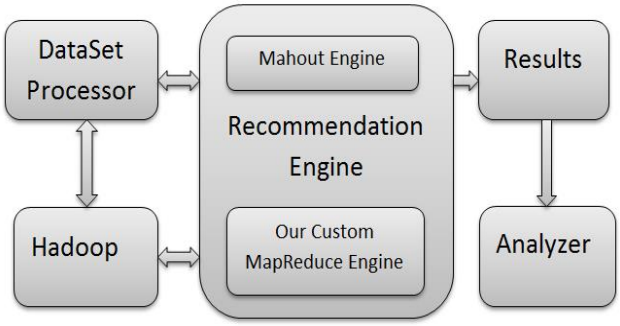

Figure 1. System Architecture

Recommendation Engine comprises *Mahout Engine* and *Custom MapReduce Engine*. These engines give top 'N' recommendations and other aggregated output. In Mahout Engine, we make use of the scalable machine-learning library whereas in custom MapReduce engine, we develop code base for Collaborative Filtering algorithms with emphasis on user-based and item-based without using external libraries or APIs.

The DataProcessor and Recommendation engine makes use of *Hadoop* MapReduce for distributed computing. Amazon AWS (EMR, EC2, S3) are used for Hadoop infrastructure.

The *Analyzer* component consists of GUI, D3 and other Analysis tools to process the results. The results (recommendations, trends and aggregated output from the Recommendation System) are reformatted in JSON format and visualized using D3.js. We also compare the recommendations from the both the engines and check the correctness and accuracy.

V. RECOMMENDATION SYSTEM DESIGN

This section describes the flow among the components of our recommendation system and also details the algorithms and techniques used in each subsystem

From Figure 2, data is processed from various datasets. The processing involves text formatting, extending dataset with other attributes and removing unnecessary information. The processing is carried out in distributed fashion for better performance using HDFS and MapReduce. This processed dataset is then loaded into HDFS file system. We use Hadoop framework and deploy our recommendation engines (Mahout and Custom MapReduce) along with data on infrastructure services like Amazon AWS (EMR, EC2, S3).

In Mahout engine, we tune the filtering parameters and use in-house algorithms to recommend movies. In our Custom MapReduce engine, we develop distributed algorithms for collaborative filtering and submit these MapReduce jobs on Amazon instances.

Job Flows are monitored and recommendations along with other aggregated output data are retrieved from HDFS. Results are again processed in the formats (JSON) required by the visualization tools (D3.js). Other movie trends, performance and statistics could be analyzed from the results.

A. *Data set Preprocessing*

The chosen data sets are Netflix and IMDB. Netflix provides a dataset containing 100480507 ratings by 480189 users for 17770 movies. Movie recommendation engine is trained on this dataset to gauge prediction success. Ratings are on a five star (integral) scale from 1 to 5. IMDB dataset has Movie Id/Name with various genre tags associated with each movie.

Dataset Formatting

Since the Netflix dataset is not favorable to process line-by-line using Hadoop framework as each movie id is at the start of the file, we format and whiten the dataset such that each line reads the format: userid, movieid, rating. We format the 17770 files in distributed manner using Hadoop MapReduce.

Merge Dataset

Netflix dataset is void of genre attribute. To augment each movie title in Netflix dataset with the list of genres, IMDB dataset and Mymovieapi.com API are used. Fuzzy Join algorithm is used for approximation movie title matching. Other attributes like location, actors, and directors may also be retrieved. To implement Fuzzy Join, we used approximate string matching algorithms like Jaccard measure and Levenshtein distance. Mix and match of Levenshtein (for smaller strings) and Jaccard (for larger strings) could be used. We also used MapReduce framework to merge datasets and to normalize (trim and upper case) the movie titles. With Movie API, we used simple HTTP GET to access information about the movies in XML or JSON format and the information included genre, actors, directors, plot, IMDB rating etc.

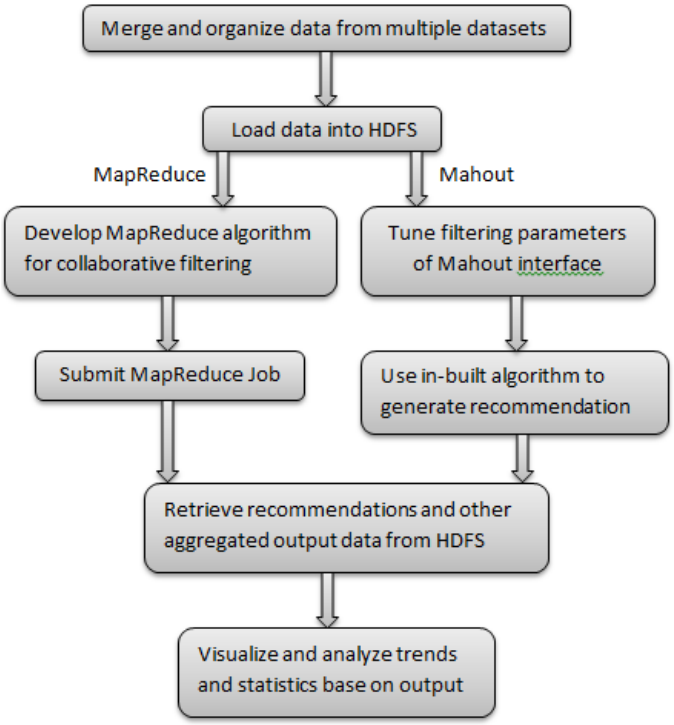

Figure 2. Recommendation System Design

B. *Map-reduce based Recommendation*

In order to recommend a product, we have to find the similar products to a product which match each other in some respects. For this, we started by calculating how similar pairs of items are. For example, in case of movie recommendations, if someone watches the movie 'Indiana Jones', the recommender would suggest the film 'Lara Croft'.

One way to find this similarity is to find the correlation between pairs of items. There are 2 challenges in finding the correlation between pairs of items:

- We are not sure that every pair of items will have some user interested in them.
- We also have to deal with case where user's interest changes with time.

Considering the above 2 challenges, we figured out that we will have some sparse data and some large data. Also, the calculation which we proceed with should be done periodically for the results to stay up to date.

1) Item-Based Collaborating Filtering MapReduce Version

In Item-Based approach, for every pair of movies A and B, we find all the people who rated both A and B. We use these

ratings to form a Movie A vector and a Movie B vector. Then, the correlation between these two vectors is calculated. Now when someone watches a movie, we can recommend him the movies most correlated with it.

In order to get a performance measure unaffected by other parameters, we developed this without using existing/pre-built packages or libraries for recommender system like Scalding or mrjob or libraries that provide functions to calculate similarity or to generate Combinations. Input requirements include User_id of movie rater, Movie_id/Name, Ratings by the users. The pseudocode of the algorithm we followed is given in Figure 3.

```
RECOM_FILT_ITEMBASED_MAPREDUCE (Ratings file):

Step 1: Map and emit (UserId, ItemId_Rating_Pair)
Step 2: For each User, Group the ItemId_Rating_Pair
          emit (UserId, (itemCount, itemRatingSum, (All itemRating pairs)))
Step 3: For Each user, Find all (item, rating) combinations
          emit (itemPair, ratingPair)
Step 4: Combine all common (itemPair, ratingPair) combinations for each user
Step 5: Find Pearson correlation similarity (All item-rating combinations)
Step 6: Sort (all the similarities for all the item-pairs)
```

Figure 3: Algorithm for Item Based Collaborating filtering

Module 1- Count_Ratings

- Mapper: For each user, emit a row containing their 'postings' (item, rating)
  Input to this Mapper class is *User id, Movie id* and *Rating* and the output is in the form of key-value pair where key is *User id* and value is (*Movie id, Rating)* pair.
- Reducer: Emit the user rating sum and count for use in later steps.
  The input to this reducer is in the form of key-value pair resulted from the mapper where key is *User id* and value is a list of (*Movie id, Rating*) pairs. The output of this reducer is also in the form of key-value pair where key is *User id* and value is *No. of ratings, Sum of all ratings* and *list of (Movie id, Rating)* pairs.

Module 2- Calculate_Similarity

- Mapper: Emits all possible combinations of movie pairs (item x, item y)
  The results from the reducer of count ratings were given to this mapper and for each user we found all the combinations of the movie ids which he has rated. The result was in the form of key-value pairs where key gives (*Movie1, Movie2*) pairs and value part stores the ratings as (*Rating1, Rating2)* pair.
- Reducer: Sum components of each pair across all users who rated both item x and item y, then we calculated pairwise similarity and co-rating counts.
  Results from Calculate_Similarity Mapper were given to this reducer in form of *Movie1, Movie2* and set of *Rating 1, Rating2* pairs which gave the similarity for every movie pair *(Movie1, Movie2)*
  The correlation can be expressed with help of Pearson's :

$$Corr(X,Y) = \frac{n\sum xy - \sum x \sum y}{\sqrt{n\sum x^2 - (\sum x)^2}\sqrt{n\sum y^2 - (\sum y)^2}}$$

Module 3- Calculate_Ranking

Sort the top correlated items for each item and print it to the output

- Mapper: Emit items with similarity in key for ranking.We gave results from Calculate_Similarity module to the Calculate_Ranking module and got the output in the form of *Movie1, Similarity* as key and *Movie2,Count of the Rating* as Value.
- Reducer: For each item emit K closest items
  We finally gave Movie1, Similarity and list of *(Movie2, Count of Ratings)* which we got from the mapper to reducer and got the result as (*Movie1, Movie2, Similarity and Count)*

2) User-Based Collaborating Filtering MapReduce Version

Implemented a map reduced version of user based collaborative filtering in java. We used movielens dataset for testing. Briefly it consists of three steps, creating user profiles for each user, find k user similarity neighborhood and recommend movies by selecting user that has more similarity. Currently, we have developed this in three modules namely 'CreateUserProfile' where we train the algorithm with sample data by creating user profiles for each user, 'KUserNeighborhood' where we calculate the correlation with other user and form a k-user neighborhood for the given user and 'RecommendationManager' which chooses one of the k-users and recommend movies. The pseudocode of the algorithm we followed for User-based CF is given in Figure 4.

```
RECOM_FILT_USERBASED_MAPREDUCE (Ratings file):

Step 1: Map and emit (ItemId, UserId_Rating_Pair)
Step 2: For each Item, Group the UserId_Rating_Pair
          emit (ItemId, (userCount, userRatingSum, (All user Rating pair)))
Step 3: For Each item, Find all (user, rating) combinations
          emit (userPair, ratingPair)
Step 4: Combine all common (userPair, ratingPair) combinations for each item
Step 5: Find Pearson correlation similarity (All user-rating combinations)
Step 6: Sort (all the similarities for all the user-pairs)
```

Figure 4: Algorithm for User Based Collaborating filtering

Module 1- User_Ratings

- Mapper: For each movie, emit a row containing their 'postings' (user, rating). Input to this Mapper class is User id, movie id and Rating and the output is in the form of key-value pair where key is Movie id and value is (User id, Rating) pair.
- Reducer: Emit the user rating sum and number of ratings for use in later steps. The input to this reducer is in the form of key-value pair resulted from the mapper where key is Movie id and value is a list of (User id, Rating) pairs. The output of this reducer is also in the form of key-value pair where key is Movie id and value is No. of ratings, Sum of all ratings and list of (User id, Rating) pairs.

Module 2- User_Similarity

- Mapper: Emits all possible combinations of movie pairs (user x, user y).The results from the reducer of count ratings were given to this mapper and for each movie we found all

the combinations of the user ids which he has rated. The result was in the form of key-value pairs where key gives (User1, User2) pairs and value part stores the ratings as (Rating1, Rating2) pair.

- Reducer: Sum components of each pair across all movies who rated the movie User 1 and User 2, then we calculated pairwise similarity and co-rating counts. Results from User_Similarity Mapper were given to this reducer in form of User1, User2 and set of Rating 1, Rating2 pairs which gave the similarity for every user pair (User1, User2)

Module 3- User_Ranking

Sort the top correlated Users for each user and print it to the output

- Mapper: Emit Users with similarity in key for ranking. We gave results from User_Similarity module to the User_Ranking module and got the output in the form of User1, Similarity as key and User2, Count of the Rating as Value.
- Reducer: For each User, emit K closest users. We gave User1, Similarity and list of (User2, Count of Ratings) which we got from the mapper to reducer and got the result as (User1, User2, Similarity and Count)

C. *Recommendation using Mahout*

The recommender system is also implemented using the existing Mahout libraries. Implementations are done for both user-based and item-based recommendations. The latest Hadoop version that is available on EMR is 1.0.3, and it contains jars for Apache Lucene 2.9.4. However, the recommender job depends on Lucene 4.3.0. So we used a bootstrap script to update the Apache Lucene 2.9.4.

1) *User Based Recommendation Implementation*

The recommender system is built using the implementations from org.apache.mahout.cf.taste.impl. These packages provide the implementations for the key abstractions like DataModel, UserSimilarity, ItemSimilarity, UserNeighbourhood, and Recommender. Mahout has built in packages only for non-distributed version. Hence we are using pseudo recommender job run the non-distributed version concurrently on different machines.

Basic architecture:

- *DataModel:* DataModel is the interface to information of user preferences. This can be implemented using FileDataModel to access data from file.
- *User / Item Similarity:* It is a representation of similarity between any two users / items
- *UserNeighbourhood:* Used as a means to determine the Neighborhood for similar users. This will be used for finding user based recommendations
- *Recommender:* It is the core abstraction in Mahout, which is used to build the recommender using other abstractions.

To create a data model, the input has to be in CSV format with lines of the form "userID, itemID, prefValue". We have used FileDataModel for the implementation. UserSimilarity is found using Pearson Correlation / LogLikelihood depending on the arguments to the programWe have then computed the UserNeighbourhood using the calculated UserSimilarity. For testing purposes we have used the Neighborhood distance as 3.

A GenericUserBasedRecommender is created using the DataModel, UserSimilarities, UserNeighbourhood.This produces output as a file which contains UserId [list of itemID] in each row.

2) *Item Based Recommendation Implementation:*

The key idea used by mahout to implement Item based recommendations is that we can ignore pairs of items without a co-occurrence rating and we need to see all co-occurrence ratings for each pair of items in the end[20].The ItemSimilarityJob class computes all item similarities. Some of the various configuration options are which similarity measure to use (e.g. cosine, Pearson-Correlation, Tanimoto-Coefficient), maximum number of similar items per item, maximum number of co-occurrences considered. The input is the preference data as CSV file, each line represents a single preference in the form userID, itemID, value and output is pairs of itemIDs with their associated similarity value.

VI. PERFORMANCE EVALUATION

A. *Experimental Setting*

We used Amazon AWS (S3, EMR, EC2) to deploy our application jar and store data. The development was carried out in local machines (Linux, OSX) and virtualized instances (Cloudera and HortonWorks). After initially testing it locally with smaller inputs for program correctness, we used to deploy and verify it on AWS. We used D3.js as our visualization tool and Twitter Bootstrap toolkit to design our website. A simple http server was created from python commands to initially deploy the website locally and later moved to Amazon instance.

Some movies had more than 10000 ratings from different users. At least ${}^{n}C_{2}$ ($n > 10000$) combinations were needed to find the co-occurrences of this movie with others. We could easily check the performance gain using AWS against local sequential processing of large dataset. To predict ratings and retrieve recommendations we deployed our code in AWS with more than 20 EMR instances.

B. *Performance Metrics*

The algorithm is targeted to recommend movies for a user based on ratings. It will also predict movie rating for a user. The other outputs include listing top rated movies for a particular period say a year/ five years or decade, listing top viewed movies, predicting top rated genre for various time ranges. Other outputs will be provided focusing on the goal of increasing the viewership and increasing advertisement sales. These outputs can be visualized using D3.js.

The performance of the implemented algorithm is compared against the built-in algorithm of the Mahout libraries. This is done using “confusion matrix” as shown in Table 1. This matrix is a kind of visualization of the performance of the algorithm. Each row of the matrix correspond to each user and the number of movies recommended by mahout library and each column corresponds to each user and represents no of movies recommended by Map Reduce version implemented.

Another confusion matrix can be built based on the movies where the values in the matrix represent number of users a movie is recommended to.

	MapReduce Based Recommendations				
Mahout Based Recommendations	Users	1	2	..	n
	1	3	0	0	0
	2	1	2	0	0
	..	0	2	0	0
	n				

Table 1. Confusion Matrix for Performance Analysis

We also used the Root Mean Square Error (RMSE) measure which represents the sample standard deviation of the differences between predicted values and observed values to check and compare the Mahout and Map-Reduce versions. We calculated the standard deviation of the Ratings using the formula : RMSE = $\sqrt{(\Sigma(\text{ActualRating} - \text{PredictedRating})^2/n)}$. RMSE for sample Netflix probe dataset with Mahout User-based was 1.3167, for Custom User-based was 1.3658, for Mahout Item-based: 1.2667 and for Custom Item-based was 1.2708.

C. *Experimental Results*

1. *Results for Data set merge:*

With equality join we got 10293/17770 movie titles which matched and with Jaccard fuzzy join (0.8 threshold), we got 10530/17770 movie titles matched.

2. *Intermediate Similarity/Correlation results*

Figures 5 and 6 show the sample intermediate results while performing Map-reduce based Collaborating Filtering.

```
User Similarities

1,7     0.005374014546466749,25
1,6     0.6396021490668314,6
10,7    0.2721655269759087,10
10,8    0.2927700218845599,16
24,16   0.8703882797784892,4
102,7   0.6657502859356826,8
11,268  0.9999999999999998,5
11,7    0.8528028654224417,8
```

Figure 5. User Similarity intermediate results

MovieA	MovieB	Correlation
Return of the Jedi (1983)	Empire Strikes Back, The (1980)	0.787655
Star Trek: The Motion Picture (1979)	Star Trek III: The Search for Spock (1984)	0.758751
Star Trek: Generations (1994)	Star Trek V: The Final Frontier (1989)	0.72042
Star Wars (1977)	Return of the Jedi (1983)	0.687749
Star Trek VI: The Undiscovered Country (1991)	Star Trek III: The Search for Spock (1984)	0.635803
Star Trek V: The Final Frontier (1989)	Star Trek III: The Search for Spock (1984)	0.632764
Star Trek: Generations (1994)	Star Trek: First Contact (1996)	0.602729

Figure 6. Intermediate Results for User Based CF showing the correlation between 2 movies.

3. *Recommendation results:*

The recommended movies for the different algorithms used are shown in Figures 7-11. Figure 7 shows the list of movieIds

```
Movie Recommendations

1       4,19,23,5,41,19,32,1,10,18
7       5,11,3,27,47,32,39,17,46,95
10      32,1,23,29,49,41,54,5,10,18
11      5,28,25,1,11,32,56,39,20,27
24      29,34,19,17,4,39,47,5,11,59
```

Figure 7. Sample intermediate results of Movie Recommendations which shows userIds and corresponding movieIds

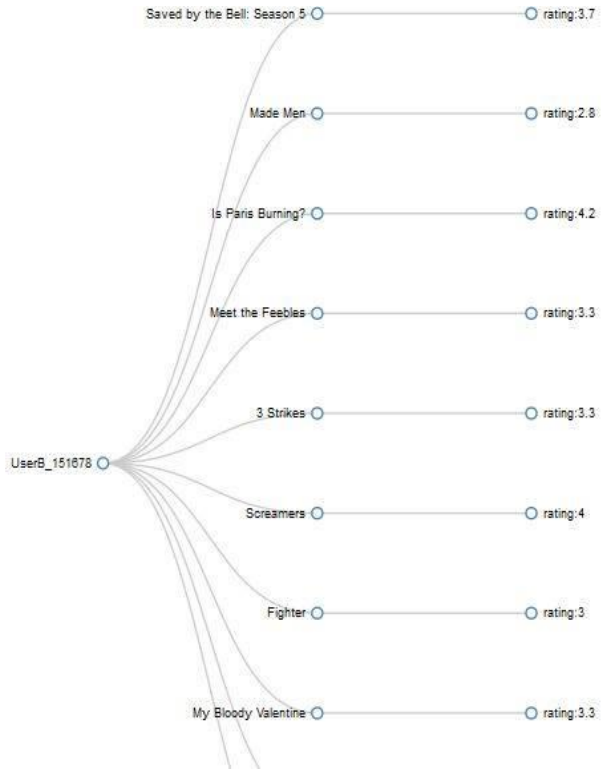

Figure 8. Top movie recommendations by Item-Based MapReduce version

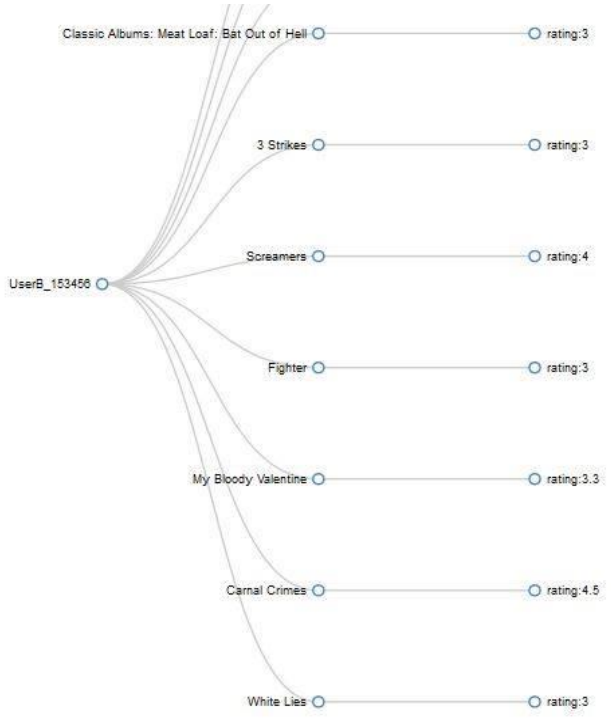

Figure 9. Top movie recommendations by User-Based MapReduce version

recommended for each user Id as a result of user-based collaborative filtering algorithm using our recommendation engine.

In Figures 8-11, the results of our recommendation system for item-based and user-based CF have been shown for both Map-Reduce and Mahout versions. We used the 'Rotating Cluster Layout' of D3.js to visualize the recommended movies in a hierarchical manner for each user.

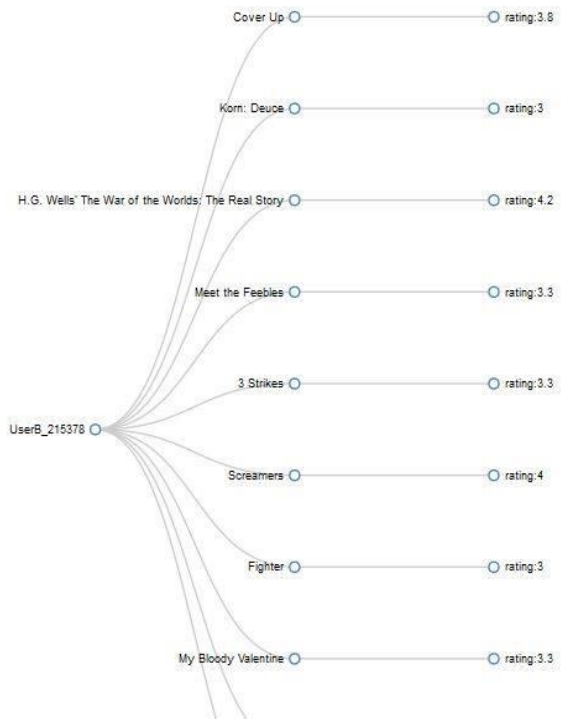

Figure 10. Top 10 movie recommendations given by Item Based CF Mahout version

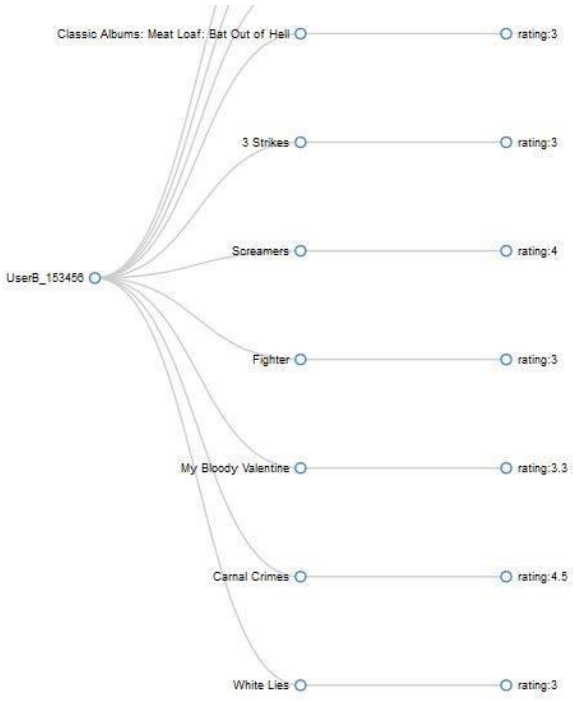

Figure 11. Top 10 movie recommendations given by User Based CF Mahout version

4. Analysis and Visualization made on Results obtained:

Based on the data obtained various similarity algorithms used, we found different data sets that we could use for further analysis. We found 'n' top rated movies for the top 'n' genres for the last n years. In the Figures 11 and 12, n is 3. For Example, as seen in Figure 12, in 2013, 'Horror' was one of the 3 top genres and in Horror, "The Conjuring" was a top rated movie.

We implemented this using Map Reduce. We have used the dataset that we obtained by merging the genre information. So one of the input files to do this prediction is file with the information movieId, year of release, movie title, list of genres that this movie falls into. Hence, we call this file as 'movieyeargenre' file. Top genres are predicted by using the rating of the movies that this particular genre has. Hence the other input file we need is the file that we generated as part of predicting recommendations using item based recommendations. This file has information regarding the movie id and its predicted rating.

To predict top 'n' genres the first step is to extend 'movieyeargenre' file with the 'movierating' file using mapper reducer. Mapper emits move id as the key and all other information as value from both the files. Reducer/combiner combines the values which has year, genre list, movie title as one the values and rating as other value with the same key i.e movie id. This information is kept in a new file called as 'yearmovieidrating' file.

The second step is to find the top genres from yearmovieidrating file. In the second Mapper reducer, mapper splits year, genre as key and rating, movieId, movie title extended with one more field called count of movies as the value. Count of movies is set to one in the map phase.The reducer combines the keys with same year and genre by adding the ratings, count of movies and all other values. So the reducer phase has year, genre as key and sum of ratings, count of movies, movieId 1, title 1, movie rating 1, movieId 2, title2, movie rating2, movieId3, title3, movie rating 3 as the values.

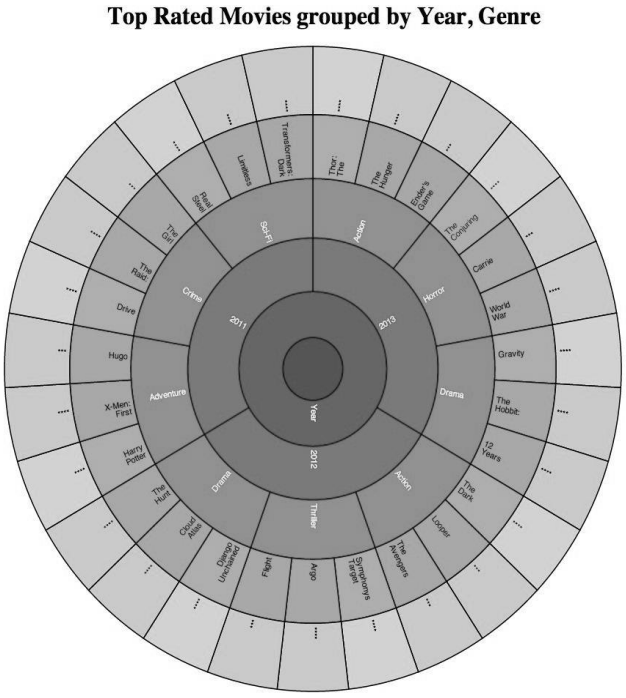

Figure 12. Top rated movies for last 3 years based on top genres. Refer Figure 13 for clarity.

In the third phase of Mapper reducer, mapper finds the average rating of the genre by using sum of ratings and count of movies and the reducer does nothing. Other information in each row is left untouched. Each row in the output file has year, genre, average rating of genre, movie id 1, movie title 1, movie id 2, movie title2, movie id 3, movie title 3, etc. The top 'n' genres are found by sorting using the average rating of genre and this information is put in the final file. This file has less than 100 records hence it will processed sequentially after this. Each record has one of the top genre for that year and the list of movie ids and movie titles and their ratings of that genre. This list is sorted by movie ratings and the top 'n' movies are picked. Hence the final prediction has top three genres for a particular year, top three movies in that genre and their ratings as depicted in the Figures 12 and 13.

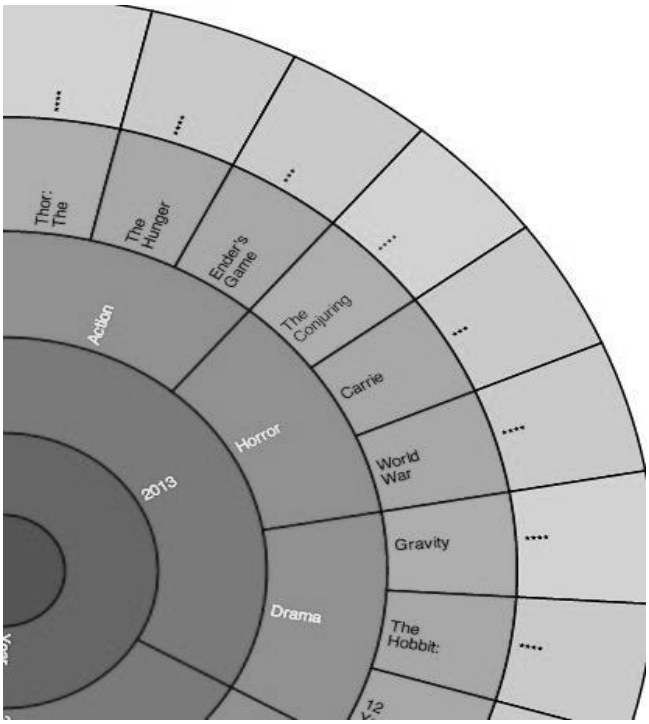

Figure 13. Final visualization used to give the ratings of the top 3 movies in the top 3 genres for last 3 years as in Figure 12

VII. CONCLUSION

In our project we built movie recommendation system using several collaborative techniques. We implemented our own distributed version of collaborative techniques using Hadoop MapReduce and compared it with that of Mahout library. The results look promising as we have comparable performance with that of Mahout.

A lot of work has been done in the field of recommendation systems, and based on available tools and libraries like Mahout there are fairly straightforward ways to build a recommender system for one's site. In addition there are many areas for experimentation using different algorithms, methods and hybrid systems to improve performance in terms of both run time and prediction accuracy. Based on the quality of the results desired for the target audience, amount of resources available and amount of manpower available in terms of development, a given engine can be adjusted in many ways to find the right balance for that particular application.

Having seen the effectiveness of these user-based and item-based collaborative techniques, it would be interesting to see how they compare to content-based and latent-based models. While many papers hinted that collaborative filtering greatly outperformed content-based algorithms, we feel there is a potential to utilize both along with latent-models for high effectiveness.

REFERENCES

[1] Aggarwal, C. C., Wolf, J. L., Wu, K., and Yu, P. S. , Horting Hatches an Egg: A New Graph-theoretic Approach to Collaborative Filtering. In Proceedings of the ACM KDD Conference, San Diego, CA, 1999.

[2] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl, "Item-Based Collaborative Filtering Recommendation Algorithms", GroupLens Research Group/Army HPC Research Center, University of Minnesota, Minneapolis.

[3] Zhi-Dan Zhao, Ming-Sheng Shang, 'User-Based Collaborative-Filtering Recommendation Algorithms on Hadoop', Third International Conference on Knowledge Discovery and Data Mining, 2010.

[4] D. Goldberg et al., 'Using Collaborative Filtering to Weave an Information Tapestry,' Comm. ACM, vol. 35, 1992

[5] J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. In OSDI'04: Sixth Symposium on Operating System Design and Implementation,San Francisco, CA, December, 2004.

[6] S. Funk, 'Netflix Update: Try This at Home,' Dec. 2006; http://sifter.org/~simon/journal/20061211.html

[7] Prem Melville, Raymond J. Mooney, and Ramadass Nagarajan. 'Content-Boosted Collaborative Filtering for Improved Recommendations'. In Proceedings of the Eighteenth National Conference on Artificial Intelligence (AAAI-2002) , Edmonton, Canada, July 2002.

[8] Sean Owen, R.A., Ted Dunning and Ellen Friedman: 'Mahout in Action' (Manning Publications, 2010. 2010)

[9] Herlocker, J. L.; Konstan, J. A.; Terveen, L. G.; Riedl, J. T. (January 2004). "Evaluating collaborative filtering recommender systems". ACM Trans. Inf. Syst.

[10] R. J. Mooney and L. Roy "Content-based book recommendation using learning for text categorization", Proceedings of the SIGIR-99 Workshop on Recommender Systems: Algorithms and Evaluation, Berkeley, CA, August 1999

[11] Tamer Elsayed, Jimmy Lin and Douglas W. Oard, "Pairwise Document Similarity in Large Collections with MapReduce", MD Proceedings of ACL-08: HLT, pp 265–268,Columbus, Ohio, USA, June 2008.

[12] Apache Mahout, http://mahout.apache.org

[13] Greg Linden, Brent Smith, and Jeremy York, Amazon.com Recommendations : 'Item-to-Item Collaborative Filtering' 2003 Published by the IEEE Computer Society 1089-7801/03/©2003 IEEE

[14] http://www.win.tue.nl/~laroyo/2L340/resources/Amazon-Recommendations.pdf

[15] http://www.amazedsaint.com/2013/07/building-simple-recommender-engine.html

[16] http://infolab.stanford.edu/~ullman/mmds/ch9.pdf

[17] http://www2.research.att.com/~volinsky/papers/ieeecomputer.pdf

[18] https://blog.twitter.com/2012/

[19] ting-recommendations-mapreduce-and-scalding

[20] https://cwiki.apache.org/confluence/display/MAHOUT/Mahout+on+Elastic+MapReduce

[21] https://github.com/mbostock/d3/wiki

[22] http://isabel-drost.de/hadoop/slides/collabMahout.pdf