Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 83 (2016) 1000 – 1006

Procedia Computer Science

International Workshop on Big Data and Data Mining Challenges on IoT and Pervasive Systems (BigD2M 2016)

A Hybrid Distributed Collaborative Filtering Recommender Engine Using Apache Spark

Sasmita Panigrahi[a]*, Rakesh Ku. Lenka[a], Ananya Stitipragyan[a]

[a]*IIIT Bhubaneswar, Bhubaneswar, Odisha, Pincode-751003, India.*

Abstract

In the big data world, recommendation system is becoming growingly popular. In this work Apache Spark is used to demonstrate an efficient parallel implementation of a new hybrid algorithm for User Oriented Collaborative Filtering method. Dimensionality reduction techniques like Alternating Least Square and Clustering techniques like K-Means are used in order to overcome the limitations of Collaborative Filtering such as data Sparsity and Scalability. We also tried to alleviate the cold start problem of Collaborative Filtering by correlating the users to products through features (tags).

Keywords: Big Data, Spark, Machine learning, Parallel Computing, Recommendation Engine, Collaborative Filtering, Hive, Hadoop

1. Introduction

In the big data world, recommendation system is becoming growingly popular. The reason is this automated tool connects the shopper with best suited products to purchase by correlating the product contents and the expressed feedback. One of the most prominent prevalent technique of Recommender Engine is Collaborative Filtering[8] [CF] .It depends only on past user actions such as past transaction or item feedback. Traditional Collaborative Filtering algorithms such as *The neighborhood approach*[13,6] and *latent factor models*[1] typically suffer from three main issues. Firstly, Cold Start[8] Problem which is basically related to the breakdown of recommenders which cannot infer preferences especially for new users for which it does not have sufficient information. Secondly, Scalability[1,4] which can be defined as the ability of Recommender of producing recommendations in real time or near to real time for very large scale datasets. Lastly, Sparsity[1] of the User-Item rating matrix as most active users will have only rated

* Corresponding author. Tel.: +91-778702381
E-mail address: sasmita239@gmail.com

doi:10.1016/j.procs.2016.04.214

few items out of all the total Items. To solving the recommendation problem, many researchers tried different approaches such as clustering[8,15] and building feature based recommender using tag[6]. Many also tried hybrid techniques[9,10]. Admittedly, however, these approaches fail for massive datasets. Recent work successfully parallelize collaborative filtering algorithms with Hadoop technology[5,7]. But Map Reduce is not computation time and cost efficient[4]. It also has not favourable scalability[4].

This work presents a new hybrid solution to user based traditional CF methods based on the Apache Spark platform[2] combining both dimension reductionality[1] and clustering methods[13] of machine learning. Also, tried to alleviate the cold start problem of Collaborative Filtering by correlating the users to products through features (tags).

A brief introduction to Apache spark is given in section 2. In Section 3, we have described the proposed work. In Section 4 we have shown parallel implementation of the work over Apache Spark. Section 5 describes experimental evaluation and result. Finally, we have concluded the findings and highlighted some future work in Section 6.

2. Introduction to Apache Spark

Spark[11,2] is an open source new big data analytics framework which solves iterative algorithms through in-memory computing created at UC Berkeley's AMPLab. It supports a much wider range of functionality than Hadoop's MapReduce[19]. The reason for the success of Spark in executing programs much faster than its counterpart Map Reduce is the use of Resilient Distributed Dataset (RDD)[2] as its programming block. RDD in Spark, an immutable distributed collection of objects, is split into multiple partitions, and then these partitions are computed on different nodes of the cluster in parallel. The new Data Frame[3] API introduced in Spark-1.4. 1 release is even performing faster than RDDs and also provides SQL like operation on RDDs. There are two types of shared variables in Spark: broadcast variables[14], which are used to store a value in memory on all nodes, and accumulators[14], which are variables which can only be "added" to, such as counters and sums. Another factor involved in the efficiency of Spark is Lazy Evaluation[2,3]. In the context of Spark, this means only *actions* are evaluated and the *transformations* are only stored for future execution. Transformations construct a new RDD from a previous one based on some condition, e.g., map, filter, etc. Actions compute a result based on an RDD, and either returns it to the driver program[11] or save it to an external storage system.

3. Proposed Work:

The proposed algorithm utilizes 20M benchmark dataset of Movie Lens[18] consisting of 20 million ratings. In order to check the scalability of the proposed algorithm we have also used 1M dataset consisting of 1 million ratings and 10M dataset consisting of 10 million ratings User can rate to a movie on a range of 1 to 5 and also can tag to a movie.

The Recommender System has two main modules, Existing User Module and New User Module as shown in figure 1 and figure 2 respectively. Data is loaded to Hive[17] and relevant features are extracted. As Preprocessing step, Users who had given less than 30 ratings and Movies which have an average rating below 3 are removed. All Tags are converted to lowercase and Stop words[6] are removed .The new preprocessed dataset is listed in below table.

Table No 1 Pro-Processing of data

Attributes	Before	After
Users	138493	110615
Movies	24744	16409
Tags	465000	441252

Once the model is built with the preprocessed dataset, it is saved to Hive in parquet format[14]. All these computations are done offline. In real time we simply load these models back from Hive which further used to generate topN recommendation. Hence, it increases the throughput of our recommender engine significantly.

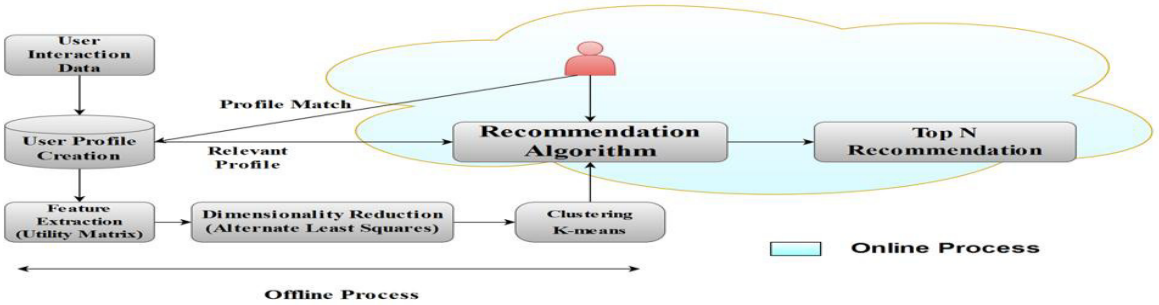

Fig 1. Block Diagram For Existing-User Recommender Module

3.1. Existing User Module

After Feature Selection at first step we build the User-Item ratings matrix. If we have M users and N movies, then the User-Item ratings matrix U is the matrix of size |M| x |N| containing all the ratings. But the matrix is very sparse which can directly affect the accuracy of the model. In this paper the Alternating Least Square method is used to overcome the Sparsity problem of existing CF by mapping the user-item matrix to a low dimensional latent factor space. This is the most widely used and served as a benchmark for CF because of its two main benefits. First, this is very easy to parallelize. Second, it works efficiently with implicit datasets. Mathematically, Our task is to find two matrices, P (|M| x K) and Q (|N| x K) such that their product approximately equals to U is given by: $U \approx P \times Q^T = U'$. P models the latent features of the users and Q models the latent features of the items. The objective is to minimize the objective function given in equation 1.

$$\min_{q^*,p^*} \sum_{(u,i)\in k} (r_{ui} - q_i^T p_u)^2 + \lambda \left(|q_{|i|}|^2 + |p_{|u|}|^2\right) \quad (1)$$

Where, q_i indicates item feature vector, p_u indicates user feature vector, λ indicates Regularization parameter, r_{ui} indicates rating given by user, u for item, i and the dot product $q_i^T p_u$ shows the interaction between the user, u and the item, i. For each iteration, the algorithm alternatively solves for the other keeping one factor matrix constant, till the values converge.[1]

K Means clustering is used to cluster similar users based on the feature set built by ALS model. K Means clustering is a paradigm of grouping items into discrete number of clusters. The Lloyd's algorithm[15], a methodology for solving the k-means clustering problem, is showcased as follows. First, we need to assume the optimal number of clusters k. The main goal of the algorithm is to minimize the objective function also called squared error function given by:

$$J = \sum_{j=1}^{k} \sum_{i=1}^{n} ||x_i^{(j)} - c_j||^2 \quad (2)$$

Where, '$||x_i - c_j||$' is the Euclidean distance between x_i and c_j, 'x_i' equals the number of data points in i^{th} cluster, 'c' shows the number of cluster centers. Users who are the most closest to their cluster center are the ones who really act as the representatives of that cluster. We termed them as the most relevant users. At first we retrieved the top k most relevant users of each cluster and saved it to Hive table.

In real time, for a user at first we find out to which cluster the user belongs to. Then all those top N highly rated movies by the relevant users of that cluster which are not seen by the user are returned as recommendation.

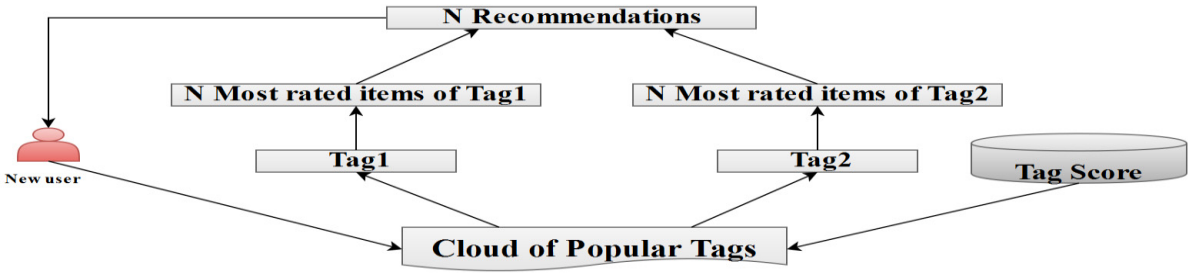

Fig 2. Block Diagram For New-User Recommender Module

3.2. New User Module:

Users can assimilate tags easily and hence tags serve as a bridge helping users to better discern an unknown relationship between an item and themselves. At first, the Tag-Score for each tag is computed. For a tag, t and movie, i the Tag-Score is defined as,

$$\text{Tag-Score}(t,i) = \frac{\text{Number of times t has been applied to i}}{\sum \text{Number of times any tag applied to i}} \quad (3)$$

The new user has to select the tags he liked from the list and on the basis of his preference the most relevant top N items related to the preferred tags are returned as recommendation to the user.

4. Parallel Implementation On Apache Spark:

Here,we will describe the implementation of our proposed work on Spark. All the algorithms are written in Scala[16] programming language. At first, we imported the rating(rating.csv) and tag file(tag.csv) file to HDFS[19].The execution of spark starts by creating a sparkContext[11] object. As data is going to be accessed repeatedly we cache it in memory. The algorithm is made up of three separate components as described in section-3: Dimension reduction, Clustering computation and Tag-Score computation. For collaborative filtering Spark's MlLib supports only one algorithm i.e., Alternating Least Square(ALS). The detail algorithm for dimensionality reduction is explained below.

Input: Rating File (rating.csv) [UserId, MovieId, Rating]
Output: UserFeature <UserId, FeatureVector>
ProductFeature<MovieId, FeatureVector>
Begin:
On each worker node do in parallel:

1. **Load** the data from rating.csv file into an RDD.
   data ← load(rating.csv)
2. **ParseRating** is a user defined function, which will split the data based on comma (',') and return an RDD of Rating class object.
   ParseRating ← map(ParseRating)
   Emit <Rating(UserId, MovieId, Rating)>
3. Store the ParseRating data in memory using **cache**()
4. **randomSplit**() the RDD into trainingRDD (80%) and testRDD (20%).
5. **Do map** on testRDD and store the first two fields into another RDD.
   test ← **map**(UserId, MovieId, Rating)
6. **Emit** <UserId, MovieId>
7. **for** i = 1 to n [n is the no.of iterations]
8. **for** j = Array (1 to m) [contains different values of 'λ']
9. **for** k = Array (1 to p) [contains different values of Rank]
10. model←**ALS.train**(trainRDD)
11. predict←model.**predict**(test)
12. Error←**map**(calculateRMSE)
13. **Emit** <Error>
    end for
    end for
    end for
14. **For** the values of j and k, which gives the least Error (RMSE), repeat step 10.
15. **Emit** <UserFeature, ProductFeature>
16. Store the results in Hive tables
    model**.saveAsParaquetFile**("ALSmodel.Paraquet")

Algorithm 1: Dimensionality Reduction

The output of the above ALS algorithm is passed as the input to next K-Means clustering algorithm. In the initialization step the User-Feature vector is broadcasted to each worker node. Then the feature vector has been normalized using the *ComputeColumnSummaryStatistics* function. It Computes column-wise summary statistics. We have used Spark MLLib's K-Means algorithm to train the model. The *computeDistance()* computes the distance of each *userFeature* vector to its *clusterCenter*. The detail description of algorithm 2 is given below.

Input: UserFeature <UserId, FeatureVector>
Output: Top N Recommendation
Begin:

1. Master **broadcasts** the user feature to all Worker nodes.
   On each Worker node, do in parallel:
   Normalise the feature vector for all users.
2. Normalise←FeatureVector.**ComputeColumnSummaryStatistics()**
3. **Emit** <mean,variance>
4. **for** i = 1 to n, n = No. of iterations
5. **for** j = 1 to k, k = No. of clusters
6. cluster = kmeans.train(UserFeatureVector)
   end for
   end for
7. **for** each UserFeature:
8. clusterId←model.**predict**(UserFeature)
9. clusterCenter←model.**clusterCenters**(clusterId)
10. distance←**computeDistance**(UserFeature,clusterCenter)
11. Join the movie ids keyed on userId from ParseRating data RDD. [Step 3 of ALS]
12. **Emit** <(clusterId),Array(UserId,MovieId)>.takeOrdered(N)
13. **For** any active user U
    [U→<(clusterId),(UserId,MovieId)>]
14. U←**map**(Top N Recommendations)
    Where top N is an user defined function which will return the topN recommendations
    14.1 **filter**() the common movie ids between U and relevant
    user set emitted from step 11.
    14.2 **Emit** <Array(movieIds)>.topN where movieIds are the
    top N highest rated movie by relevance
    end for

Algorithm 2: Clustering

For the new user module, once the user selects the tags, most relevant items are returned as recommendation based on the tag score as described in section-3. Step, step, step3 mentioned in algorithm 1 is performed by function *ParseTag.DF()* for tags.csv file. The RDDs is converted to data Frame11 with the *SQLContext* Object. Spark allows to run SQL queries over the data by registering a data frame as table, which can be done using the command *dataFrame.registerTempTable("tablename").* The detail steps involved are described in the algorithm 3.

Input: tags.csv [UserId, MovieId, Tag]
Output: <UserId, MovieId, Tag, tagScore>
Begin:

1. On each Worker node, do in parallel:
   Repeat step 1 to 3 of ALS algorithm on input.
2. dataFrame←**ParseTag.DF()**
3. dataFrame.**registerTempTable**("tag")
4. val orderedId = sqlContext.sql("**SELECT** movieid **AS** id, tag **FROM** tag **ORDER BY** movieid")
5. val eachTagCount = orderedId.**groupBy**("id,tag")**.count**()
6. val finalresult = sqlContext.sql("**SELECT** movieid, tagname, occurrence **AS** eachTagCount, count **AS** totalCount **FROM** result **ORDER BY** movieid")
7. val tagScore = sqlContext.sql("**SELECT** movieid, tagname,(eachTagCount/totalCount) **AS** tagScore **FROM** finalresult"))

Algorithm 3: Computing Tag Score.

5. Experimental Evaluation and Results

All the experiments were performed on Ubuntu 14.04 operating system running on 2.50GHz processors with 4 processing cores. The master node of a cluster was allocated 4GB RAM while each slave node was allocated 2GB RAM. We used the latest released Apache Hadoop-2.7.2, Apache Hive 2.0, Spark-1.6.0, Scala -2.11.7 and SBT-0.13.9 for the purposes of all the experiments.

For the new user module we have run the algorithm-3 described in section 4 and found that to produce 10 number of recommendation a two node cluster is taking only 0.67seconds. To find the approximate optimal values, of the

two hyper parameters of the ALS model, Rank and λ(Regularisation Parameter), we train the model over Ranks having range of {10, 50, 70, 100} and a λ range of {0.01, 0.1, 1, 10} on the 80% training partition. As shown in **Fig-5** we determine rank of 50 and λ of 0.1where the RMSE[12] is optimal, i.e., 0.88. The resultant RMSE improves upon the base model by 17%. By using the best model, we computed the RMSE value on test set and found that both the RMSE are reasonably equal. This indicates the accuracy of the model.Before applying algorithm-2 we removed the outliers of the userFeature set because it can greatly affect the accuracy of clustering results. Z-score method is used for normalization. All data falls into a range of [-1, 1]. One of the greatest challenges is to decide how many clusters(K) to make. However, a good rule of thumb is to use the "elbow method." To examine this, we have simply evaluated for a range of K = {2, 3, 4, 5, 6, 7, 8, 9, 10, 20, 30} and collect the results for WSSSE[1].(With in Set Sum Error). As per **Fig. 6**, we found that for k=20, WSSSE is minimum, i.e., 1641. 11. The running time of our algorithm is measured as the number of nodes and data size increases as shown in the **Fig. 7**. The pseudo distributed mode fails to process the 10m and 20m dataset where as the one node cluster takes 40 minutes of time to process the 10m but failed for 20m. Surprisingly, where the number nodes increased from one to two the computing time reduced drastically. The two node cluster takes only 7.59 minutes for the 20m dataset. **Fig-8** demonstrates an comparison of our model with all other standard CF algorithms with respect to throughput which is nothing but number of recommendation generated per minute. With the increase of number of clusters throughput can more be increased, however the computation time will be slightly increased.

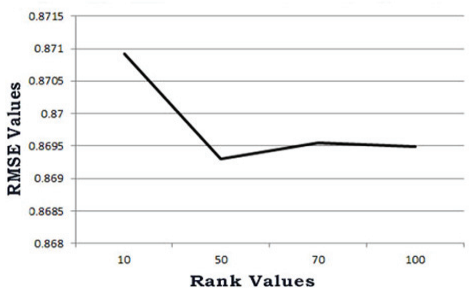

Fig 5. Impact of Rank values on RMSE.

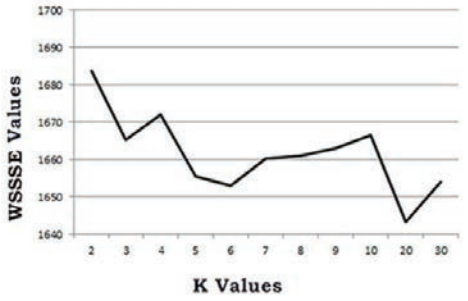

Fig 6. Determination of optimal value of K.

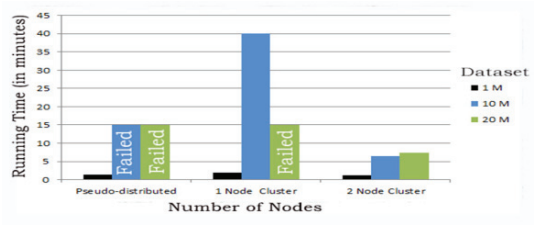

Fig 7. Runtime with increase of nodes and data size

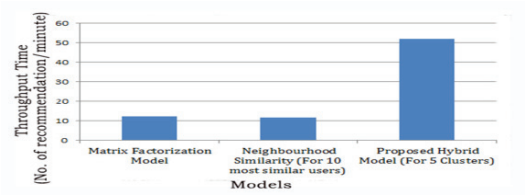

Fig 8. Comparison of all models on basis of throughput

The following table gives a detailed comparison of our models with the standard algorithms.

Table No. 2 Comparison of different models on basis of various recommendation parameters

	Matrix Factorization Model	**Neighbourhood Model**	**Proposed Hybrid Model**
Cold Start Problem	Yes	Yes	No
Scalability	Low	Least	Most
Throughput	Low	Least	Most
Sparsity	No	Yes	No

5. Conclusion & Future Work

Our model is evaluated on 1 million, 10 million and 20 million user preferences collected from Movielens. The experimental findings show that running time of the algorithm is improved with every addition of a node into Spark Cluster. Also in terms of throughput our model is giving the best result as compared to standard algorithms. Further we also include a detailed review of advantages and disadvantages of all CF algorithms in practice and found that our model performs the best among all. However few challenges we faced is Spark demands a higher RAM size for in memory computation which is expensive. For speeding up computational time, we choose Spark's native language Scala. Learning Scala programing language was initially challenging, but its functional programming features, less verbose codes makes it worth learning. However, for better prediction result we need to update the ALS model and Clusters manually. Hence the future work could be to replace K-Means with Streaming K-Means which can automatically update the model each time the chosen number of new users or new items added. We are also planning to test our model with increasing nodes on much larger data sets.

References

1. Y. Zhou, D. Wilkinson, R. Schreiber, and R. Pan, Large-scale parallel collaborative filtering for the Netflix prize, Berlin, Heidelberg, *Springer-Verlag*, In AAIM '08, pages 337– 348, 2008.
2. Sp Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. Franklin, S. Shenker, and I. Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing, *Technical Report UCB/EECS-2011-82*, EECS Department, University of California, Berkeley, 2011
3. Michael Armbrust, et al, "Spark SQL: Relational Data Processing in Spark", in Proceedings of Association for Computing Machinery, Inc. ACM 978-1-4503-2758, Pages 1383-1394, May 27, 2015
4. M. Isard and Y. Yu. Distributed data-parallel computing using a high-level programming language. Pages 987-994 In SIGMOD, 2009.
5. Z.-D. Zhao and M.-S. Shang, "User-based collaborative-filtering recommendation algorithms on hadoop," in Knowledge Discovery and Data Mining, 2010. WKDD'10. Third International Conference on. IEEE,2010, pp. 478–481.
6. S. Golder and B. A. Huberman. The structure of collaborative tagging systems. Journal of Information Science, vol. 32 no. 2 198-208, April, 2006
7. J. Jiang, J. Lu, G. Zhang, and G. Long, "Scaling-up item-based collaborative filtering recommendation algorithm based on hadoop," in Services (SERVICES), 2011 IEEE World Congress on. IEEE, 2011,pp. 490–497.
8. Adomavicius, Gediminas, and Alexander Tuzhilin. "Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions." *Knowledge and Data Engineering, IEEE Transactions on* 17.6 (2005): 734-749.
9. Burke, Robin. "Hybrid web recommender systems." In *The adaptive web*, pp. 377-408. Springer Berlin Heidelberg, 2007.
10. Ali Kohli, Seyed javad Ebrahimi and Mehrdad Jalali, "Improving the Accuracy and Efficiency of Tag Recommendation System by Applying Hybrid Methods," 2011 1st International eConference on Computer and Knowledge Engineering (ICCKE), pp 242-248, October 13-14,2011.
11. Spark Programming Guide - Spark 1.6.0 Documentation, http://spark.apache.org/docs/latest/programming-guide.html
12. Asela Gunawardana and Guy Shani , "A Survey of Accuracy Evaluation Metrics of Recommendation Tasks", Journal of MachineLearning Research , Vol. 10, pp. 2935-2962, 2009.
13. Kantor PB, Rokach L, Ricci F, Shapira B. Recommender systems handbook. Springer; 2011.
14. Holden Karau, Andy Konwinski, Patrick Wendell, and Matei Zaharia, Learning Spark - Lightning-Fast Data Analysis, O'Reilly Publications, 2015
15. Ungar LH, Foster DP. "Clustering methods for collaborative filtering". In AAAI workshop on recommendation systems (Vol. 1, pp. 114-129), Jul 26 1998
16. Scala. http://www.scala-lang.org
17. Apache Hive. http://hadoop.apache.org/hive
18. http://grouplens.org/datasets/movielens/
19. Jeffrey Dean and Sanjay Ghemawat, "Map-reduce: Simplied Data Processing on Large Clusters", in Proc. of OSDl, 2004, pp.137-150.
20. Ungar LH, Foster DP. "Clustering methods for collaborative filtering". In AAAI workshop on recommendation systems (Vol. 1, pp. 114-129), Jul 26 1998