A Fast and Better Hybrid Recommender System Based on Spark

Jiali Wang, Hang Zhuang, Changlong Li, Hang Chen, Bo Xu, Zhuocheng He, Xuehai Zhou

A Fast and Better Hybrid Recommender System Based on Spark

Jiali Wang, Hang Zhuang, Changlong Li, Hang Chen, Bo Xu,
Zhuocheng He, and Xuehai Zhou

University of Science and Technology of China, No.96 Jinzhai Road,
Hefei, Anhui, China,
{ustcwjl, zhuangh, liclong, hangc, xubo245, orcking}@mail.ustc.edu.cn,
xhzhou@ustc.edu.cn

Abstract. With the rapid development of information technology, recommender systems have become critical components to solve information overload. As an important branch, weighted hybrid recommender systems are widely used in electronic commerce sites, social networks and video websites such as Amazon, Facebook and Netflix. In practice, developers typically set a weight for each recommendation algorithm by repeating experiments until obtaining better accuracy. Despite the method could improve accuracy, it overly depends on experience of developers and the improvements are poor. What worse, workload will be heavy if the number of algorithms rises. To further improve performance of recommender systems, we design an optimal hybrid recommender system on Spark. Experimental results show that the system can improve accuracy, reduce execution time and handle large-scale datasets. Accordingly, the hybrid recommender system balances accuracy and execution time.

Keywords: Recommender System, Hybrid, Weight, Spark

1 Introduction

Along with the popularization of the Internet, a sharp increase in the amount of data leads to information overload [1]. Thus, recommender systems [2] were proposed to relieve the stress of massive data. To improve recommender systems performance, researchers put forward the weighted hybrid method. Despite performance boost has been brought by the method, there are still several problems affecting performance, including weight setting and computation load. Hence, we implement a weighted hybrid recommender system on Spark. In the system, we design a new method to compute weights, using cluster analysis and user similarity. Besides, the execution time can be reduced by deploying the system on Spark.

1.1 Hybrid Recommender Systems

Hybrid recommender systems combine two or more recommendation algorithms to overcome weaknesses of each algorithm. It is generally classified as Switching, Mixed, Feature Combination, Meta-Level, and Weighted [3].

The weighted hybrid technique combines different algorithms with different weights [3]. The main idea is that the algorithm with better accuracy has a higher weight. At present, developers always set a weight for an algorithm manually and repeat experiments until achieving superior accuracy. Thus, the method depends on developers' experience to determine accuracy of an algorithm in different datasets. Due to large-scale datasets, sparsity of rating data and the number of algorithms, it's generally hard to obtain appropriate weights. Eventually the improvements of accuracy are poor.

In addition, to improve user experience, the system should return recommendation results efficiently. In other words, it has to quickly locate information which can appeal users in massive data. Thus, execution time is another evaluation standard of performance. However, the weighted hybrid technique needs to execute two or more algorithms and compute hybrid results, it's tough to reduce execution time.

Apart from accuracy and execution time of the system, scalability is also an important consideration. With the increasing of data scale and the algorithm complexity, the system requires more storage space and computing resources. It's difficult to meet the actual demand by only optimizing algorithms.

To address the above-mentioned issues, we design a hybrid recommender system on Spark. In the system, we propose an optimized method to improve accuracy. It computes weights and hybrid results based on cluster analysis and user similarity. Meanwhile, we deploy the system on Spark which is a fast and general engine for large-scale data processing [4] to accelerate the training process and improve scalability.

1.2 Work of Paper

The rest of this paper is organized as five sections. Section 2 reviews recommendation algorithms and introduces the Spark. Section 3 describes the design of the optimized method. Section 4 shows how we implement the system on Spark. Section 5 gives experimental results and our analysis. Section 6 presents our conclusions and future work.

2 Related Work

In this section, we first review and compare recommendation algorithms and recommender systems. Then, we briefly analyze predicting ratings of algorithms. Finally, we introduce the distributed computing platform Spark and compare Hadoop and Spark.

2.1 Recommender Systems

Recommendation algorithms are the basis of recommender systems. In this section, we first introduce several representative algorithms.

Collaborative recommendation is almost the most popular algorithm. Based on overlapped ratings, it computes similarities among users. And then, it uses similarities to predict the rating that the current user on an item [5]. Tapestry [6], Ringo [7] and GroupLens [8] are typical systems with the algorithm.

Content-based recommendation pays attention to connections between items. It analyses descriptions of items that have been rated by users [9] and calculates similarities between items. The represention of an item's feature and the way to classify a new item are two important sub-problems [9].

Demographic-based recommendation [9] is a simple algorithm. It focuses on types of users that like a certain item. The technique identifies features of users such as age, gender, nationality, education, etc. It measures user similarity by taking those features into consideration. Table 1. shows strengths and weaknesses of each algorithm [5][9].

Table 1. Strengths and weaknesses of recommendation algorithms

Algorithm	Strength	Weakness
Collaborative	Field independence. Not necessary to understand descriptions of items. Support users to discover potential interests.	New user problem. New item problem. Sparsity.
Content	Improve accuracy by increasing dimensions of item features.	Cold start problem. Similarity measurement is one-sided.
Demographic	Historical data are not necessary. Wide range of applications. No cold start problem.	The algorithm is rough and imprecise.

As the simple and effective technique, the weighted hybrid recommender system has been widely used in numerous fields. P-Tango and Pazzani are two typical systems. P-Tango is an online news system. It combines collaborative and content-based recommendation algorithms. The system adjusts weights of algorithms in the process of operation. Until the system obtains the expected accuracy, it determines weights. Pazzani is the other weighted hybrid recommender system. It combines collaborative, content-based and demographic-based recommendation algorithms. The system uses voting to determine recommendation results.

2.2 Weight Analysis

As previously described in section 1, we give the formalized representation of the weighted hybrid technique as follows:

$$\widetilde{R_{ui}} = \sum_{j=1}^{n} \alpha_j r_{ui}^j \tag{1}$$

where j represents the j'th algorithm, it ranges from 1 to n. α_j corresponds to the weight of the j'th algorithm. r_{ui}^j is the predicting rating of user u on item i by the j'th algorithm. $\widetilde{R_{ui}}$ indicates the final hybrid result. From the formula (1), we can recognize that each algorithm just has a certain weight. That means the technique presupposes that predicting ratings of an algorithm are all greater or less than their ratings. However, this condition evaluates to false. Here we give some empirical evidence. We implement the User-based Collaborative Filtering (User-CF) and the Alternating Least Squares (ALS) in Python2.7, and use MovieLens-100K as observed data.

Table 2. The results of statistic analysis on predicting ratings

Algorithm	countH	countL	countE
User-CF	9181	10752	11
ALS	6992	12952	0

In the Table 2, countH is the number of predicting ratings which are greater than real ratings. The countL is less than real ratings and countE is equivalent amounts. From the empirical results, we know that:

(1) In these algorithms, there are little predicting ratings that equal to ratings.
(2) A part of predicting ratings are greater than ratings, and another are less than ratings.
(3) Only a weight for an algorithm may affect accuracy.

Thus, it is essential to optimize weights.

2.3 Spark

Spark is a fast and general-purpose cluster computing platforms for large-scale data processing [4] which is developed by UC Berkeley. In the environment of Spark, it includes Spark SQL [10], Spark Streaming [11], Mllib [12], GraphX [13], etc. Based on resilient distributed dataset (RDD) [14], it achieves memory-based computing, fault tolerance and scalability. Currently, Spark is deployed in Amazon, ebay and Yahoo! to process large-scale datasets.

For a hybrid recommender system, performance is affected by data scale, the number of algorithms and the complexity of algorithms. Deploy the system on Spark can mitigate above affects.

(1) In the system, large-scale datasets could be stored in distributed storage.
(2) Algorithms are independent with each other, they are supposed to be performed in parallel.
(3) Intermediate values can be cached in memory to decrease execution time.

Therefore, in this paper, we design an optimized hybrid recommender system on Spark.

3 Design Overview

The empirical evidence from section 2 suggests that accuracy still has chance to be improved. The predicting ratings are higher or lower than corresponding ratings. Thus, we use cluster analysis to obtain more accurate weights. The principle of cluster analysis is that according to the properties of samples, using mathematical methods to determine relationship between samples, and according to the relationship to cluster samples. Based on cluster analysis, we present an optimized method for calculating personalized weights. Now let us discuss the method in detail.

3.1 Objective Function.

In this section, we first give explanations of several concepts. In the following statement:

1. Assume that there are n algorithms in the system and j is the j'th algorithm.
2. u for user, i for item and (u,i) represents the data item of u and i.
3. R_{ui} is the rating of u on i, r_{ui}^j is the predicting rating of u on i which is computed by the j'th algorithm.
4. For the j'th algorithm, the error between the rating and the predicting rating is: $D_{ui}^j = R_{ui} - r_{ui}^j$. In order to reduce $\sum_{j=1}^{n} \sum_{u,i} D_{ui}^j$, similar errors are expected to get same weights. Based on errors, we divide (u,i) into k clusters and design $\boldsymbol{C_{ui}} = (c_1, c_2, \cdots, c_k)$ to reflect the cluster of (u,i). For the j'th algorithm, $\boldsymbol{\alpha_j} = (\alpha_{j1}, \alpha_{j2}, \cdots, \alpha_{jk})$ represents k weights of the algorithm. $\boldsymbol{\alpha_j C_{ui}^T}$ finally determines the weight for r_{ui}^j.

According to our analysis, we define the objective function as formula (2):

$$F(\boldsymbol{\alpha}) = \sum_{u,i} (R_{ui} - \boldsymbol{\alpha_1 C_{ui}}^T {r_{ui}}^1 - \boldsymbol{\alpha_2 C_{ui}}^T {r_{ui}}^2 - \cdots - \boldsymbol{\alpha_n C_{ui}}^T r_{ui}^n)^2 \qquad (2)$$

$$s.t. \sum_{j=1}^{n} \boldsymbol{\alpha_j C_{ui}^T} = 1 \qquad (3)$$

3.2 Weight Calculation

According to D_{ui}^j, the optimized method classifies all (u,i) into k clusters . For each (u,i), it has a vector $\boldsymbol{C_{ui}} = (c_1, c_2, \cdots, c_k)$ and is initialized to $\boldsymbol{C_{ui}} = (0, 0, \cdots, 0)$. The value which corresponds to (u,i)'s cluster is set to 1. For instance, if (u,i) belongs to the cluster 2, $\boldsymbol{C_{ui}} = (0, 1, 0, \cdots, 0)$. The weight for r_{ui}^j is α_{j2} which is computed by $\boldsymbol{\alpha_j C_{ui}}^T$. Therefore, $\boldsymbol{C}$ could map weights to predicting ratings and achieve multiple weights for an algorithm. Fig.1 shows the pipeline of the method.

After calculating $\boldsymbol{C}$, the optimized method requires to compute $\boldsymbol{\alpha_j}$. For the purpose of minimizing the objective function, we make use of the Lagrange theory and minimum theory [15][16]. Based on formula (2), the method constructs the Lagrange function $L(\boldsymbol{\alpha})$.

$$Ł(\boldsymbol{\alpha}) = F(\boldsymbol{\alpha}) + \lambda \sum_{u,i} \phi(\alpha) \tag{4}$$

$$\phi(\boldsymbol{\alpha}) = \sum_{j=1}^{n} \boldsymbol{\alpha_j} \boldsymbol{C_{ui}}^T - 1 \tag{5}$$

For each j, let $\frac{\partial L}{\partial(\boldsymbol{\alpha_j} \boldsymbol{C_{ui}}^T)} = 0$. We can get an equation:

$$2 * \sum_{u,i} (\boldsymbol{\alpha_1} \boldsymbol{C_{ui}}^T r_{ui}^1 r_{ui}^j + \boldsymbol{\alpha_2} \boldsymbol{C_{ui}}^T r_{ui}^2 r_{ui}^j + \cdots + \boldsymbol{\alpha_n} \boldsymbol{C_{ui}}^T r_{ui}^n r_{ui}^j) + \lambda = 2 * \sum_{u,i} R_{ui} r_{ui}^j \tag{6}$$

The equation (6) can be represented by matrix:

$$\begin{aligned} XY &= 2 * \begin{pmatrix} \boldsymbol{\alpha_1} & \boldsymbol{\alpha_2} & \cdots & \boldsymbol{\alpha_n} & \lambda \end{pmatrix} \\ &\quad * \begin{pmatrix} \sum_{u,i} \boldsymbol{C_{ui}}^T r_{ui}^1 r_{ui}^1 & \sum_{u,i} \boldsymbol{C_{ui}}^T r_{ui}^1 r_{ui}^2 & \cdots & \sum_{u,i} \boldsymbol{C_{ui}}^T r_{ui}^1 r_{ui}^n & \sum_{u,i} \boldsymbol{C_{ui}}^T \\ \sum_{u,i} \boldsymbol{C_{ui}}^T r_{ui}^2 r_{ui}^1 & \sum_{u,i} \boldsymbol{C_{ui}}^T r_{ui}^2 r_{ui}^2 & \cdots & \sum_{u,i} \boldsymbol{C_{ui}}^T r_{ui}^2 r_{ui}^n & \sum_{u,i} \boldsymbol{C_{ui}}^T \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \sum_{u,i} \boldsymbol{C_{ui}}^T r_{ui}^n r_{ui}^1 & \sum_{u,i} \boldsymbol{C_{ui}}^T r_{ui}^n r_{ui}^2 & \cdots & \sum_{u,i} \boldsymbol{C_{ui}}^T r_{ui}^n r_{ui}^n & \sum_{u,i} \boldsymbol{C_{ui}}^T \\ 1 & 1 & \cdots & 1 & 0 \end{pmatrix} \\ &= 2 * \begin{pmatrix} \sum_{u,i} R_{ui} r_{ui}^1 \\ \sum_{u,i} R_{ui} r_{ui}^2 \\ \vdots \\ \sum_{u,i} R_{ui} r_{ui}^n \\ \sum_{u,i} 1 \end{pmatrix} = R \end{aligned} \tag{7}$$

Thus the weight matrix X can be calculated by

$$X = R * Y^{-1} \tag{8}$$

The optimized method uses ratings which have already stored in the system to compute weights. However, these weights aren't entirely appropriate for a new (u,i). We further introduce user similarity to compute weights. The user similarity is computed by cosine similarity:

$$sim_{u,v} = \frac{|N_{(u)} \cap N_{(v)}|}{\sqrt{|N_{(u)}||N_{(v)}|}} \tag{9}$$

where $sim_{u,v}$ is the similarity between u and v. $N_{(u)}$ means the number of items that u have rated. $N_{(v)}$ is the same as $N_{(u)}$. For the $(u^{'}, I^{'})$, the optimized

method calculates the hybrid result as:

$$\hat{r_{u'I'}} = \frac{\sum_v sim_{u',v} * (\boldsymbol{\alpha_1} \boldsymbol{C_{vI'}}^T r^1_{u'I'} + \boldsymbol{\alpha_2} \boldsymbol{C_{vI'}}^T r^2_{u'I'} + \cdots + \boldsymbol{\alpha_n} \boldsymbol{C_{vI'}}^T r^n_{u'I'})}{\sum_v sim_{u'v}} \tag{10}$$

The equation (10) is able to filter the interference of non similar weights and get a personalized weight for the $(u^{'}, I^{'})$.

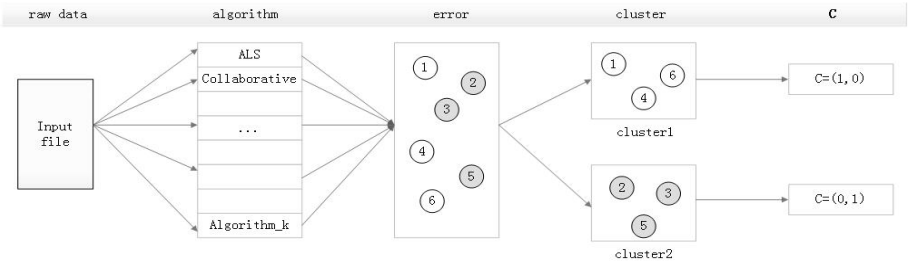

Fig. 1. The pipeline of the optimized method. The input file consists of ratings. Algorithms read the input file and output predicting ratings. Then the system computes errors and cluster data items. Finally the system gives the $\boldsymbol{C}$.

4 Implementation

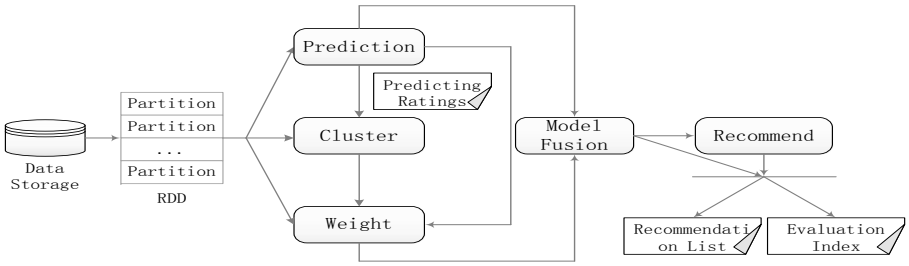

Fig. 2. The architecture of the hybrid recommender system. The system reads ratings and outptu recommendation lists. Besides, it also provides an evaluation index.

According to the design overview, we deploy the hybrid recommender system on Spark. The system contains data storage, prediction, cluster, weight, model fusion and recommendation, totally 6 modules. Fig.2 shows the architecture of the system.

4.1 Modules

Data storage module is the basis of the system. It stores input data, including historical data and ratings. We use HDFS which is a distributed file system to store raw data [17]. The pre-processed data are put in the database such as HBase, Redis, Hive, etc [18][19][20]. Topside modules read data from the database. Prediction module is used to compute predicting ratings. It performs recommendation algorithms in parallel. Outputs are predicting ratings.

The cluster module concentrates on errors of (u,i). It exploits k-means to classify (u,i). Output of the module is $\boldsymbol{C}$. The weight module accepts $\boldsymbol{C}$ to compute weights. With $\boldsymbol{C}$ and $\boldsymbol{\alpha}$, the module can get a weight for each r_{ui}^{j}. Output of it is $\boldsymbol{\alpha}$.

The model fusion calculates hbrid results based on predicting ratings, $\boldsymbol{C}$, $\boldsymbol{\alpha}$ and user similarity. According to these parameters, it determines hybrid results by logistic regression [21]. Recommendation is used to recommend items for users. Based on hybrid results, it generates recommendation lists. Besides, it also outputs an evaluation for results.

4.2 Discussion

In the hybrid recommender system on Spark, data are translated into RDDs. Because of the characteristics of memory-based computing and parallel operations, RDDs can be processed in parallel to reduce execution time. The read-only and fault tolerance of RDD make the system more reliable. Besides, due to the distributed storage of Spark, the system is able to handle large-scale datasets. It improves scalability of the system. Therefore, deploy the hybrid recommender system on Spark could decrease execution time and further improve scalability.

5 Performance

5.1 Evaluation Index

Accuracy. The system accuracy is measured by root mean square error (RMSE) [22]. It is defined as:

$$RMSE = \sqrt{\frac{\sum_{u,i \in T} (R_{ui} - \hat{r_{ui}})^2}{|T|}} \tag{11}$$

where R_{ui} and $\hat{r_{ui}}$ is the rating and the hytbrid result that u on i respectively. $|T|$ denotes the number of $\hat{r_{ui}}$.

Execution Time. The execution time includes time of algorithms, clustering, calculating weights and hybrid results. It is measured in minutes.

5.2 Experimental Setup

In this experiment, we choose Spark as our platform. All experiments were performed using a local cluster with 7 nodes (1 master and 6 worker nodes): each node has Xeon(R) dual-core 2.53GHz processor and 6GB memory.

Dataset. In Table 3, we list datasets that were used in the experiment. For each dataset, we divide it into 2 training sets and a test set randomly.

Table 3. Datasets in the experiment

Dataset	Users	Items	Ratings
MovieLens-100K	1000	1700	100000
MovieLens-200K	1371	10153	200000
MovieLens-300K	2004	10850	300000
MovieLens-400K	2661	11634	400000
MovieLens-500K	3462	13257	500000
MovieLens-600K	4073	13488	600000
MovieLens-700K	4753	14154	700000
MovieLens-800K	5543	14230	800000
MovieLens-900K	6207	14963	900000
MovieLens-1M	6000	4000	1 million
BookCrossing	71212	176272	400000

Algorithms. We implement 3 recommendation algorithms: User-CF, Item-based Collaborative Filtering (Item-CF) and ALS. We perform them in training sets and test sets to compute predicting ratings, weights and hybrid results.

Nodes. We compare execution time of the stand-alone system and the distributed system. For the former, we use the server with Xeon(R) dual-core 2.53GHz processor and 6GB memory. For the latter, we use a local cluster with 7 nodes (1 master and 6 worker nodes): each node has Xeon(R) dual-core 2.53GHz processor and 6GB memory.

5.3 Performance Comparision

In this section, we evaluate performance of the hybrid recommender system on Spark, including accuracy and execution time.

Fig.3 shows impacts of data scales on accuracy. In the experiment, we performe the combination of User-CF and ALS on four MovieLens datasets. In the Fig.3, with the increasing of data scale, RMSE generally decreases. Due to the

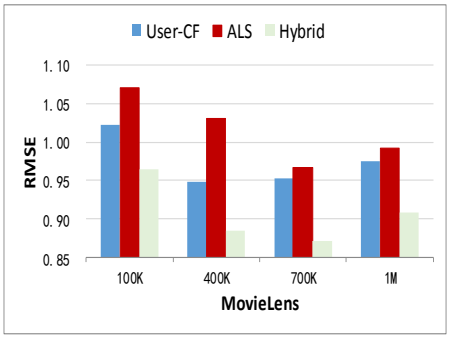

Fig. 3. The RMSE of different scale MovieLens. The x-axis indicates datasets, and the y-axis describes the RMSE.

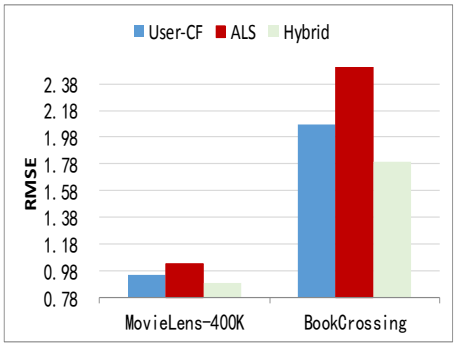

Fig. 4. The RMSE of different types of datasets. The x-axis indicates datasets, and the y-axis describes the RMSE.

sparsity of MovieLens-700K, the hybrid recommender system obtains the best result. Compare with User-CF and ALS, the system improves accuracy of 8.21

Fig.4 gives the RMSE of different types of datasets. In the experiment, we performe the combination of User-CF and ALS on MovieLens-400K and BookCrossing. The Fig.4 shows that the hybrid recommender system can improve accuracy of different types of datasets. And there are significant improvements on BookCrossing. The improvements demonstrate that the system is available for sparse datasets.

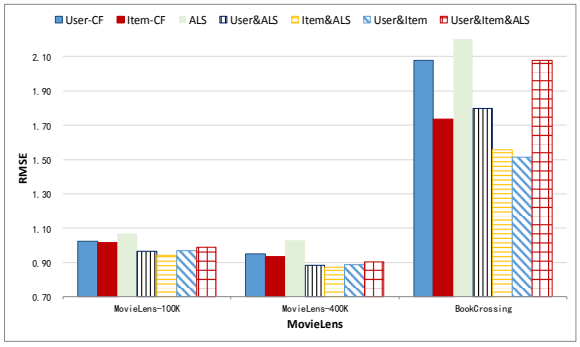

Fig. 5. The RMSE of different combinations of algorithms. The x-axis indicates the dataset, and the y-axis describes the RMSE.

Fig.5 shows correlations between accuracy and combinations of algorithms. In the experiment, four combinations of algorithms are performed on MovieLens-100K, MovieLens-400K and BookCrossing respectively. In Fig.5, the hybrid recommender system obtains better accuracy than single algorithm. When accura-

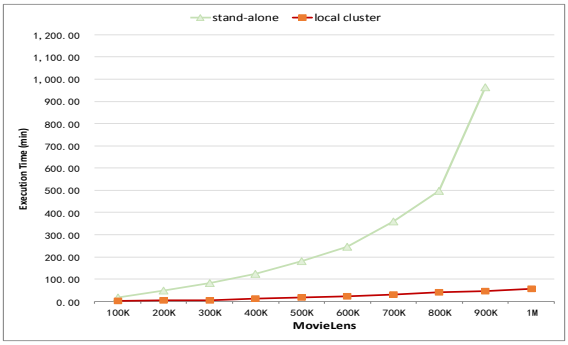

Fig. 6. The execution time of 2 modes. The x-axis indicates datasets and the y-axis describes execution time.

cy of single algorithm is favorable, the hybrid recommender system also obtains better accuracy.

Fig.6 compares execution time of stand-alone mode and local cluster mode. The experiment performs the combination of User-CF and ALS on MovieLens-100K to MovieLens-1M. For the stand-alone system, execution time increases sharply with the expansion of data scale. However, execution time of local cluster mode remains relatively constant. When the data scale is larger than MovieLens-900K, the stand-alone mode couldn't handle it. The local cluster mode could handle MovieLens-10M or larger datasets. From Fig.6, we can recognize that memory-based computing, parallel operations and distributed storage of Spark are helpful to decrease execution time and improve scalability.

6 Conclusion and Future Work

Improving performance of recommender systems is a crucial solution for information overload. This paper designs a new weighted hybrid recommender system to solve this problem. We are the first to compute weights by using cluster analysis, user similarity and minimum theory. Besides, we deploy the hybrid recommender system on Spark. The system improves accuracy by optimizing weights and reduces execution time by memory-based computing and parallel operations. And distributed storage of the system is helpful to improve scalability. The experiment results demonstrate the performance of our hybrid recommender system.

In future work, we will consider to improve and extend the system: expansion of algorithm to process more complex scenes. Further research on factors influencing weights to improve accuracy. Meanwhile, optimize the implementation of the system on Spark.

References

1. Eppler, M.J., Mengis, J.: The concept of information overload: A review of literature from organization science, accounting, marketing, mis, and related disciplines. The information society **20**(5) (2004) 325–344
2. Cosley, D., Lam, S.K., Albert, I., Konstan, J.A., Riedl, J.: Is seeing believing?: how recommender system interfaces affect users' opinions. In: Proceedings of the SIGCHI conference on Human factors in computing systems, ACM (2003) 585–592
3. Burke, R.: Hybrid recommender systems: Survey and experiments. User modeling and user-adapted interaction **12**(4) (2002) 331–370
4. Zaharia, M., Chowdhury, M., Franklin, M.J., Shenker, S., Stoica, I.: Spark: Cluster computing with working sets. HotCloud **10** (2010) 10–10
5. Burke, R.: Hybrid systems for personalized recommendations. In: Intelligent Techniques for Web Personalization. Springer (2005) 133–152
6. Goldberg, D., Nichols, D., Oki, B.M., Terry, D.: Using collaborative filtering to weave an information tapestry. Communications of the ACM **35**(12) (1992) 61–70
7. Shardanand, U., Maes, P.: Social information filtering: algorithms for automating word of mouth. In: Proceedings of the SIGCHI conference on Human factors in computing systems, ACM Press/Addison-Wesley Publishing Co. (1995) 210–217
8. Resnick, P., Iacovou, N., Suchak, M., Bergstrom, P., Riedl, J.: Grouplens: an open architecture for collaborative filtering of netnews. In: Proceedings of the 1994 ACM conference on Computer supported cooperative work, ACM (1994) 175–186
9. Pazzani, M.J.: A framework for collaborative, content-based and demographic filtering. Artificial Intelligence Review **13**(5-6) (1999) 393–408
10. Armbrust, M., Xin, R.S., Lian, C., Huai, Y., Liu, D., Bradley, J.K., Meng, X., Kaftan, T., Franklin, M.J., Ghodsi, A., et al.: Spark sql: Relational data processing in spark. In: Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, ACM (2015) 1383–1394
11. Zaharia, M., Das, T., Li, H., Shenker, S., Stoica, I.: Discretized streams: an efficient and fault-tolerant model for stream processing on large clusters. In: Presented as part of the. (2012)
12. Meng, X., Bradley, J., Yavuz, B., Sparks, E., Venkataraman, S., Liu, D., Freeman, J., Tsai, D., Amde, M., Owen, S., et al.: Mllib: Machine learning in apache spark. arXiv preprint arXiv:1505.06807 (2015)
13. Xin, R.S., Gonzalez, J.E., Franklin, M.J., Stoica, I.: Graphx: A resilient distributed graph system on spark. In: First International Workshop on Graph Data Management Experiences and Systems, ACM (2013) 2
14. Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauley, M., Franklin, M.J., Shenker, S., Stoica, I.: Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In: Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation, USENIX Association (2012) 2–2
15. Borneas, M.: On a generalization of the lagrange function. American Journal of Physics **27**(4) (1959) 265–267
16. Mitra, N.J., Nguyen, A.: Estimating surface normals in noisy point cloud data. In: Proceedings of the nineteenth annual symposium on Computational geometry, ACM (2003) 322–328
17. Borthakur, D.: The hadoop distributed file system: Architecture and design. Hadoop Project Website **11**(2007) (2007) 21

18. Zhang, D.W., Sun, F.Q., Cheng, X., Liu, C.: Research on hadoop-based enterprise file cloud storage system. In: Awareness Science and Technology (iCAST), 2011 3rd International Conference on, IEEE (2011) 434–437
19. Han, J., Haihong, E., Le, G., Du, J.: Survey on nosql database. In: Pervasive computing and applications (ICPCA), 2011 6th international conference on, IEEE (2011) 363–366
20. Thusoo, A., Sarma, J.S., Jain, N., Shao, Z., Chakka, P., Anthony, S., Liu, H., Wyckoff, P., Murthy, R.: Hive: a warehousing solution over a map-reduce framework. Proceedings of the VLDB Endowment **2**(2) (2009) 1626–1629
21. Tsukimoto, H.: Logical regression analysis: from mathematical formulas to linguistic rules. In: Foundations and Advances in Data Mining. Springer (2005) 21–61
22. Willmott, C.J., Matsuura, K.: Advantages of the mean absolute error (mae) over the root mean square error (rmse) in assessing average model performance. Climate research **30**(1) (2005) 79