Pipeline Item-based Collaborative Filtering based on MapReduce

Zhi-Lin Zhao*, Chang-Dong Wang*‡, Yuan-Yu Wan*, Zi-Wei Huang* and Jian-Huang Lai†‡
*School of Mobile Information Engineering, Sun Yat-sen University, Zhuhai, P. R. China, 519082
†School of Information Science and Technology, Sun Yat-sen University, Guangzhou, P. R. China, 510006
‡SYSU-CMU Shunde International Joint Research Institute (JRI), Shunde, P.R. China, 528300
Email: zhaozhl7@mail2.sysu.edu.cn, changdongwang@hotmail.com, wanyy6@mai2.sysu.edu.cn,
niuian0524@gmail.com, stsljh@mail.sysu.edu.cn

***Abstract*—As we all know, it is an era of information explosion, in which we always get huge amounts of information. Therefore, it is in urgent need of picking out the useful and interesting information quickly. In order to solve this serious problem, recommendation system arises at the historic moment. Among the existing recommendation algorithms, the item-based collaborative filtering recommendation algorithm is the most widely used one. Its principle is based on the user's evaluation of items. The purpose is to find the similarity between users, and recommend items to the target user according to the records of the similar users. However, the number of customers and products keeps increasing at a high rate, which increases the cost to find out the recommendation list for each user. The efficiency of a single common computer will not satisfy the requirement and the super computer will cost too much. In order to solve the problem, we propose to use MapReduce to implement the recommendation system. Besides, we distribute the job to some computer clusters and the input file of the current computer cluster only relies on the previous one or the origin input. So the pipeline technology will be adopted to improve the efficiency further. The experiment shows that the method can merge the ability of some common PC to process large-scale data in a short time.**

***Index Terms*—item-based collaborative filtering; recommendation; MapReduce; pipeline**

I. Introduction

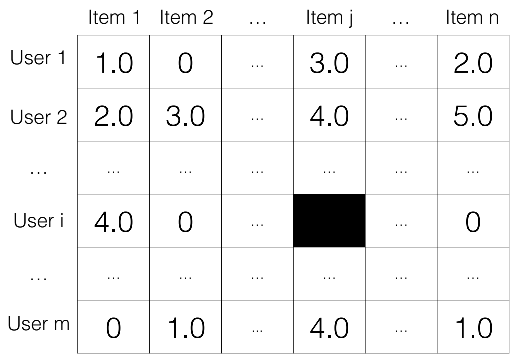
In the era of information explosion, recommendation system [1] is one of most useful tools to help users get the interesting information in a short time. Recommendation system is based on the historical records of user access, purchasing records and relation between items to construct interest model, and with the user interest model of multifarious information, finally the system will recommend users the items they might be interested in. The collaborative filtering [2]–[6] is one of the most popular techniques used in recommendation system. One of the earliest item-based collaborative filtering recommendation systems was developed in [7], which used the similarities of items to make recommendations to users.

Due to the continuous development of the global information industry, it becomes more and more difficult for network resource and data analysis technology to adapt to the demand in present era of the intensive data processing. In recent years, the cloud computing emerges. It is a kind of resource on-demand renting service mode, where users can access the data in the computer and storage system according to the requirement. The network of the computer resources, virtual as a resource pool, and the specific software are used to realize automatic and intelligent computing. So it allows a variety of computing resources to work together. Based on this technology, Google Labs put forward MapReduce model in the cloud computing, which is specialized in parallel computation of large data sets. In our project, we realize a pipeline item-based collaborative filtering by using MapReduce. The map operations on the input data can calculate an intermediate key/value pairs. Properly combining the data, we apply reduce operation in all the value which have the same key. The use of function model, combined with the user to specify the map and reduce operations, so we can easily realize the large-scale parallel computing. At the same time we use the restart mechanism which can easily realize fault tolerance. Like MapReduce, this parallel computing model of analyzing large data sets has formed a "cluster" revolution in the industry. The current software implementation is to specify a map function, using a set of keys for mapping into a new set of key/value pairs, specifying the concurrent reduce function, to ensure that all the keys of the map for each of the group share the same key.

There are some researches about recommendation algorithm running in a distributed cloud computing, but them still exist some issues to be addressed. In [8], a MapReduce framework was proposed to calculate the similarities of any two points, but failed to implement the entire recommendation system on MapReduce platform. In [9], a framework was constructed to implement the recommendation system on MapReduce. However, it is limited to only one cluster. When applied in more than one clusters, it will fail to make full use of computing resources since some of the clusters are wasted.

In the paper, we propose a pipeline [10] item-based collaborative filtering recommendation algorithm built on four Hadoop clusters. For each cluster, it receives the input from the previous cluster and provides the input for the next cluster so that we can run the recommendation algorithm in the pipeline model. One cluster can begin a new job after finishing the task of the whole recommendation system and it's unnecessary for the cluster to wait for the accomplishment of the whole job which can improve the utilization of those computing clusters.

The algorithm we proposed is more efficient and can take



	Item 1	Item 2	…	Item j	…	Item n
User 1	1.0	0	…	3.0	…	2.0
User 2	2.0	3.0	…	4.0	…	5.0
…	…	…	…	…	…	…
User i	4.0	0	…	■	…	0
…	…	…	…	…	…	…
User m	0	1.0	…	4.0	…	1.0

Fig. 1. User-item rating matrix

advantage of multiple clusters to complete recommendation system with the growth of data scale. It make a contribution to fill the lack of recommendation algorithm running in a distributed cloud computing.

Our experiment shows that the method we proposed can merge the ability of some common PC to process large-scale data in a short time.

II. Preliminaries

In this section, we will briefly introduce the problem definition and some notations used in this paper.

First of all, we need to construct a user-item rating matrix from the original data, as shown in Fig. 1.

Let $U = \{u_1, u_2, \ldots, u_m\}$ be the collection of users. The number of users is m. Let $I = \{i_1, i_2, \ldots, i_n\}$ be the collection of items. The number of items is n. Let R be the user-item rating matrix. And R_{ij} is the rating user i gives to item j. If the user i don't rate item j, the value of R_{ij} is set to zero.

When calculating the similarity of items, we just consider adjusted cosine similarity. So the similarity between items i and j is given by

$$S_{ij} = \frac{\sum_{u \in U}(R_{ui} - \bar{R}_u)(R_{uj} - \bar{R}_u)}{\sqrt{\sum_{u \in U}(R_{ui} - \bar{R}_u)^2}\sqrt{\sum_{u \in U}(R_{uj} - \bar{R}_u)^2}} \quad (1)$$

where $\bar{R}_u$ is the average of the ratings of user u.

We think that the prediction of rating given by user u to item i is equal to the sum of the weighted ratings given by the user on the items similar to i. The weight is the corresponding similarity between i and other item.

Therefore, let S_{ij} be the similarity between i and j, the predicted rating user u give to item i, denoted as p_{ui}, can be calculated as follows:

$$p_{ui} = \frac{\sum_{j=0}^{k} S_{ij} * R_{uj}}{\sum_{j=0}^{k} |S_{ij}|} \quad (2)$$

Intuitively, the higher the predicted rating of the item is, the more possible the target user is willing to buy. For the target user, we sort all the predicted ratings and recommend the items with high predicted ratings for the user.

III. Item-based Collaborative Filtering Recommendation Algorithm based on MapReduce

In order to execute the Item-based Collaborative Filtering Recommendation algorithm on MapReduce, we should divide the algorithm into four steps and the outputs of the previous MapReduce operation are the inputs of the succeeding MapReduce operation. The major goal of the first two steps is to get all the items' similarity, and after that the rest steps can predict the scores of the items which the target user have not bought yet. In this section, we will describe the four steps in detail and show the pseudocode of each step.

A. Step one

In this step, we just prepare the data for the second step to calculate the similarity of each pair of items.

1) Mapper I: The goal of the first map operation in the first map machine is to sent all the $< Item, Rating >$ pairs from the same user to the same reduce machine so that we can get the user's average rating score in the reduce machine. Each input value is a rating record $MIA = UserID :: ItemID :: Rating$.

INPUT :

$$< UserID :: ItemID :: Rating >$$

OUTPUT :

$$< UserID, (ItemID : Rating) >$$

Algorithm 1 Mapper I

```
1: Input: $MIA$.
2: [$UserID$, $ItemID$, $Rating$] = Split $MIA$ by '::'
3: $key$ = $UserID$
4: $value$ =$ItemID$ : $Rating$
5: Output: ($key$ ,$value$)
```

2) Reducer I: Since all the records of a user have been shuffled to the same reduce machine, the average rating score $Rating$ of the user should be calculated firstly. After that, we can get the score difference DR of all the items' rating to the average rating score which make it possible for us to use the Adjusted Cosine Similarity method to get all the item similarities later. In fact, it is unnecessary for us to get all items permutations but to make the ID of the previous item large than the later one in the item pair because the items similarities matrix we just need to construct is symmetric.

INPUT :

$$< UserID, list(ItemID : Rating) >$$

OUTPUT :

$$< (ItemID_i : ItemID_j), (DR_i : DR_j) >$$

B. Step two

All the item similarities will be calculated in this phase and only $\frac{n*(n-1)}{2}$ item similarities will be listed in the end since the items similarities matrix is symmetric.

Algorithm 2 Reducer I

```
1: Input: $key$,$value[]$.
2: $UserID$ = $key$
3: $len$ = length($value[]$)
4: $ItemID[len]$
5: $Rating[len]$
6: let $L$ be an empty list
7: for $i$ = 1 to $len$ do
8:     [$ItemID_i$,$Rating_i$] = Split $value_i$ by '::'
9: end for
10: $ave$ = The mean of Rating[]
11: for $i$ = 1 to $len$ do
12:     for $j$ = $i+1$ to $len$ do
13:         $DR_i$ = $Rating_i$ - $ave$
14:         $DR_j$ = $Rating_j$ - $ave$
15:         $t$ = $Item_i$ : $Item_j$, $DR_i$:$DR_j$
16:         L.push(t)
17:     end for
18: end for
19: Output: $L$
```

1) Mapper II: The map machine will do nothing in this phase because the output of the first reduce machine can be the input of the reduce machine in the second step directly. $B = (ItemID_i : ItemID_j), (DR_i : DR_j)$ is the input of the map machine.
INPUT :

$$< (ItemID_i : ItemID_j), (DR_i : DR_j) >$$

OUTPUT :

$$< (ItemID_i : ItemID_j), (DR_i : DR_j) >$$

Algorithm 3 Mapper II

```
1: Input: $MIB$.
2: [$key$, $value$] = Split $MIB$ by ','
3: Output: ($key$ ,$value$)
```

2) Reducer II: For two items i and j where ($i < j$), all the co-rating pairs from the same user will be shuffled to the same reduce machine. In other words, all the corresponding value pairs (DR_i, DR_j) coming from the users who have rated the item i as well as item j will be sent to the same reduce machine. Notice that the corresponding value of the item i or j in the machine is the difference between the item's rating and the average rating of the user to all the items. In this way, we can use the Adjusted Cosine Similarity method to get the similarity between the item i and the item j.
INPUT :

$$< (ItemID_i : ItemID_j), list(DR_i : DR_j) >$$

OUTPUT :

$$< (ItemID_i : ItemID_j), S_{ij} >$$

Algorithm 4 Reducer II

```
1: Input: $key$,$value[]$.
2: $len$ = length($value[]$)
3: for $i$ = 1 to $len$ do
4:     [$DRA_i$,$DRB_i$] = Split $value[t]$ by ':'
5: end for
6: $S$ = $\frac{DRA*DRB}{|DRA|*|DRB|}$
7: $ItemIDs$ = $key$
8: Output: ($ItemIDs$,$S$)
```

C. Step three

There are two sources for the input in this step. One is the input of the first step and the other one is the output of the second step. Different operations will be carried out to the different input streams in the map machine. In order to put the item similarities and user rating records in the same reduce machine, all the results will take the same key in the key/value pair after the map operation.

1) Mapper III: This is the most important phase in the entire algorithm. There are two kinds of output from two different sources respectively. In order to distinguish the two different types of output obtained from corresponding inputs, tag should be attached to each key/value pair. MIC1 = $UserID :: ItemID :: Rating$ is the input of the first Mapper as well as the type A input of the third Mapper. MIC2 = $ItemIDs, S$.
INPUT :

$$< UserID :: ItemID :: Rating >$$
$$< (ItemID_i : ItemID_j), S_{ij} >$$

OUTPUT :

$$< ItemID_i, (A : UserID : Rating_i) >$$
$$< ItemID_i, (B : ItemID_j : S_{ij}) >$$
$$< ItemID_j, (B : ItemID_i : S_{ij}) >$$

Algorithm 5 Mapper III (Type A)

```
1: Input: $MIC1$.
2: [$UserID$, $ItemID$, $R$] = Split $MIC1$ by '::'
3: $key$ = $ItemID$
4: $value$ =$A : UserID : R$
5: Output: ($key$ ,$value$)
```

2) Reducer III: The reduce machine will combine the result from different sources in the map machine. So the reduce machine will also have two kinds of input with the same key but different tags, namely tag A or tag B. The key $ItemID_i$ of A and B is a bridge to connect two kinds of input. The value of type A input tuple is the user who has rated item i and the corresponding rating while the value of type B input tuple is the item j that the user in type A value tuple has not bought yet and the similarity between item i and item j.
INPUT :

Algorithm 6 Mapper III (Type B)

```
1: Input: $MIC2$
2: [$ItemIDs$,$S$] = Split $MIC2$ by ','
3: [$ItemID_i$, $ItemID_j$] = Split $ItemIDs$ by ':'
4: $key1$ = $ItemID_i$
5: $value1$ =$B$:$ItemID_j$:$S$
6: $key2$ = $ItemID_j$
7: $value2$ =$B$:$ItemID_i$:$S$
8: $out[1]$ = ($key1$,$value1$)
9: $out[2]$ = ($key2$,$value2$)
10: Output: (out)
```

$$< ItemID_i, list(A : UserID : Rating_i) >$$
$$< (ItemID_i : ItemID_j), S_{ij} >$$

OUTPUT :

$$< UserID : ItemID_j, Rating_i : S_{ij} >$$

Algorithm 7 Reducer III

```
1: Input: $RICA$, $RICB$.
2: $RICA$ is the output lists of Mapper A
3: $RICB$ is the output lists of Mapper B
4: $lenA$ = length($RICA$)
5: $lenB$ = length($RICB$)
6: let $L$ be a list
7: for $i$ = 1 to $len(RICA)$ do
8:     for $j$ = 1 to $len(RICB)$ do
9:         UserID,R = Split $RICA_i$ by ':'
10:        ItemID,S = Split $RICB_i$ by ':'
11:        t = UserID:ItemID, R:S
12:        L.push(t)
13:    end for
14: end for
15: Output: ($L$)
```

D. Step four

In step four, we will get the prediction rating of the all items the target user has not bought yet.

1) Mapper IV: The output of step 3 like this $(UserID : ItemID_j, Rating_i : S_{ij})$ and the output value with the same $(UserID : ItemID_j)$ will appear in many reduce machines. The task of the map machine in this phase is to collect those data and sent the key/value pair with the same $(UserID : ItemID_j)$ as key to the same reduce machine by shuffle sort so that we can predict the user's rating to item j. MID = $UserID : ItemID_j, Rating_i : S_{ij}$ is the input of the fourth map machine.

INPUT :

$$< UserID : ItemID_j, Rating_i : S_{ij} >$$

OUTPUT :

$$< UserID : ItemID_j, Rating_i : S_{ij}) >$$

Algorithm 8 Mapper IV

```
1: Input: $MID$.
2: [$key$,$value$] = Split $MID$ by ','
3: Output: ($key$, $value$)
```

2) Reducer IV: For $item_j$, we can get a gain from each item i that the user u has rated before by $R_{ui} * S_{ij}$. The sum of all the gains into the sum of all the corresponding similarities is the prediction score to the item j.

INPUT:

$$< UserID : ItemID_j, list(Rating_i : S_{ij}) >$$

OUTPUT :

$$< UserID : ItemID_j : PredictScore_j >$$

For the target user, we can calculate the $PredictScore_j$ the predicted rating of item j by

$$PredictScore_j = \frac{\sum_{i=1}^{k} Rating_i * S_{ij}}{\sum_{i=1}^{k} |S_{ij}|} \quad (3)$$

where k is the number of items that the user has rated and the k items will be put into the same reduce machine.

Algorithm 9 Reducer IV

```
1: Input: $key$,$value[]$.
2: $len$ = length(value[])
3: $SumA = 0$
4: $SumB = 0$
5: for $i$ = 1 to $len$ do
6:     ItemID,S = Split $value[i]$ by ':'
7:     $SumA = SumA + ItemID * S$
8: end for
9: $PredictScore$ = $SumA$ / $SumB$
10: $UserID$,$ItemID$ = Split $key$ by ':'
11: Output: ($UserID$,$ItemID$,$PredictScore$ =)
```

The algorithm we design can be divided into four steps. There are a certain number of computers for map operator and reduce operator in each step. Because the time complexity of the four steps are different, we can balance the execution time of the four steps by controlling the number of computers of four MapReduce computing clusters. These four successive steps are not in a serial manner but like an assembly line as shown in Fig. 2. The input files will not be sent to Step Two until Step One finishes processing them. Step One will continue to deal with new input information as Step Two is dealing with the previous input files in good order. Working like an assembly line, this model can increase the efficiency, accelerate the processing of data as well as reduce the costs. Notice that, the third Hadoop cluster need two input files, one is the original input file and the other one is the output of the second Hadoop cluster. In order to implement pipeline model, we should make some modification in the first step and the second step. The first Hadoop cluster should copy the original input and sent its original input and the output of the first step to the second Hadoop cluster. It means that the second map machine and reduce machine will receive two kinds of input source just like the third Hadoop. We can use the same way used in the third map machine and reduce machine(using two different tags) to distinguish the two kinds of inputs. The first Reducer should give a tag to each output so that the second Mapper can distinguish them. When the second Hadoop receive the two kinds of outputs with different tags from the first Hadoop cluster it can recognize them and execute corresponding operation. In the way, the third Hadoop can receive two different inputs from the output file of the second Hadoop cluster directly. But if we just have one Hadoop and don't want to implement pipeline, those modification in the first step and the second are not necessary and we just need to implement the code according to the pseudocode given before. Getting the original input by sending the original input file again to the third step directly is better than getting from the output file of the second step after transfering between the first and the second step because the transmission time can be saved.

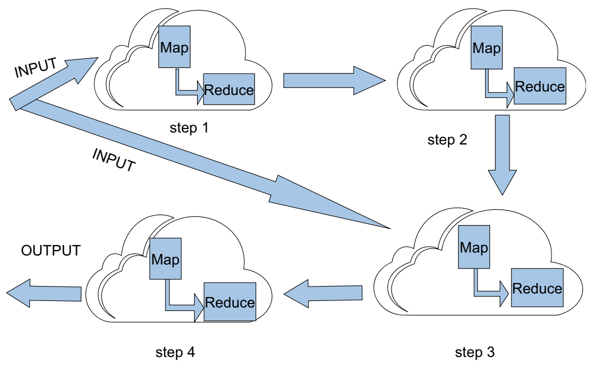


Fig. 2. Pipeline MapReduce model

IV. Experiments

In this section, we evaluate the efficiency of the item-based collaborative filtering algorithm on the Hadoop cluster. As expected, the prediction results obtained by the the item-based collaborative filtering algorithm on the Hadoop cluster are the same as the algorithm executing on the single computer. The efficiency of the algorithm on Hadoop cluster will be improved as the number of the computers of the Hadoop cluster increasing.

A. Experimental environment

We evaluate the efficiency of the algorithm on a Hadoop cluster which consists of 6 nodes. One node is mater node which is used to manage the whole Hadoop cluster and the other five nodes are data notes that is in charge of processing data distributed by the mater node. All the nodes are common personal computers with Intel i3-3240 CPU, 8G RAM, 150G disk, running Ubuntu Linux 14.04. All the machines are connected by switch.

TABLE I
MovieLen Datasets

MovieLen					
#Users	1040	3870	6710	9730	12802
#Movies	6728	8925	9574	9914	10068
#Ratings	126804	505818	888775	1317295	1729636

B. Experimental dataset

In this experiment, we use the MovieLens dataset [11] which collects users' rating records from the website. The dataset contains 71567 users, 65133 movies and 10000054 ratings on the range from 1 to 5. In order to evaluate the speed improvement obtained by our method, we extract five different datasets with different sizes from the original large dataset. The five datasets are shown in the Table I. For convenience, we name the five datasets according to their sizes, i.e., "1040 users dataset", "3870 users dataset", etc.

C. Experimental evaluation

We use the evaluation measurement which is proposed in [12] to evaluate the efficiency of the item-based collaborative filtering algorithm on the Hadoop cluster. The number of node represent the number of the computers in the Hadoop cluster.

$$f = \frac{T_1}{T_p} \tag{4}$$

where T_1 is the execution time with one node in the Hadoop cluster and T_p is the execution time with p nodes on the Hadoop cluster. So we can know the influence of the number of the nodes to the execution efficiency via the value of f.

Ideally, the execution time of the algorithm will decrease linearly as the number of nodes on the Hadoop cluster increases. The data transmission between the nodes on the Hadoop cluster will cost some times. We support the transmission time is α and the execution time with p nodes is not equal to $\frac{T_1}{p}$ but $\frac{T_1}{p} + \alpha$. The larger the data size is (which implies that the longer execution time with one node will be), the larger the α value will be. Therefore $\alpha \propto T_1$. Now, we can rewrite Eq. 4 as,

$$f = \frac{T_1 * p}{T_1 + \alpha * p} \tag{5}$$

The experimental results on the Hadoop cluster with different nodes and different scale of datasets are shown in Fig. 3. As the number of nodes increases in the Hadoop cluster, the execution time of the algorithm will be reduced. If we ignore the transmission time between each node, the execute speed is p times the original speed where the p is the number of nodes in the Hadoop cluster. Notice that the improvement on the larger dataset will be more obvious because the smaller dataset will have a higher percentage of the data transmission time from the total but the main cost of the bigger dataset is to process the data in each node. The execution time of the four steps on the dataset with 12802 users are displayed in Fig. 4. The four steps of a job will run in the cluster with the same number of nodes. As we can see in the figure, the second step will cost more than the other three steps but all the four steps can be accelerated as the number of nodes on the Hadoop cluster increasing. In order to run the whole algorithm with pipeline, we should balance the execution time of the four steps by controlling the number of nodes in each Hadoop cluster. We should put more nodes in the second Hadoop cluster since it will spend more time finishing its task. For the whole algorithm, if the running time of the four steps are almost the same, we can run the algorithm in the pipeline model. It is expected that a step finishes its task, the succeeding step can finish its task at the same time and get the next input file from the previous step immediately. In this way, we can make all the nodes on all the Hadoop clusters busy to speed up execution.

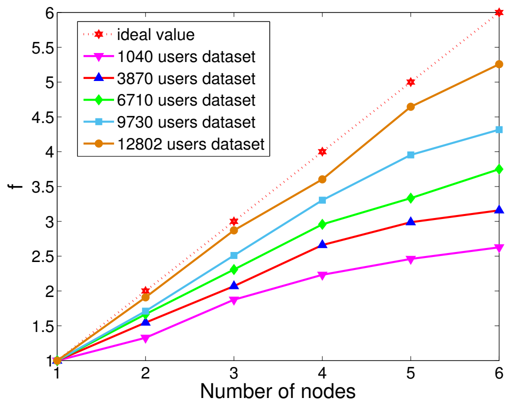


Fig. 3. The efficiency of Hadoop cluster

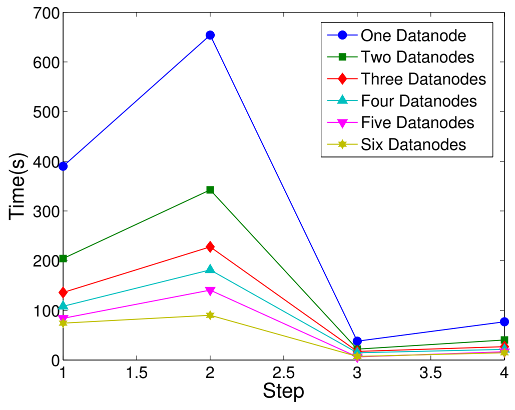


Fig. 4. The time-consuming of the four steps

V. Conclusions

Recommendation system is a very popular topic and it has brought great convenience to our daily life. With the development of science and technology, recommendation system was applied in more and more situations. But with the growth of data, recommendation system needs more time and power to meet the requirements. There should be a new scheme to solve the problem.

In this paper, we execute the traditional item-based collaborative filtering recommendation algorithm on MapReduce by splitting the whole big dataset into some smaller datasets. In this way, we can build a powerful computing cluster by collecting the computing ability of many common PCs to process a large scale dataset efficiently. If we have four Hadoop clusters, we can run the whole algorithm in pipeline which can make good use of all the computing resources by making all the computers busy.

Our results show that the proposed method shortens the execution time of recommendation system. But there are still many work we should do in the future. In this paper, we just propose a method to execute the traditional item-based collaborative filtering on Hadoop. As the development of the recommendation system, the quality of the recommendation algorithm will be improved but the algorithm may be more complex.

Acknowledgment

This project was supported by NSFC (61173084 and 61128009), CCF-Tencent Open Research Fund, the PhD Start-up Fund of Natural Science Foundation of Guangdong Province, China (No. 2014A030310180), the Fundamental Research Funds for the Central Universities (46000-3161006), Pilot Program of SYSU-CMU Shunde International Joint Research Institute, and Undergraduate Innovation Program of Sun Yat-sen University.

References

[1] M. D. Ekstrand, J. T. Riedl, and J. A. Konstan, "Collaborative filtering recommender systems," *Human–Computer Interaction*, pp. 84–87, 2011.
[2] A. Das, M. Datar, and A. Garg, "Google news personalization: Scalable online collaborative filtering," in *WWW 2007/Track: Industrial Practice and Experience*, 2007, pp. 271–280.
[3] J. Wang and J. Yin, "Combining user-based and item-based collaborative filtering techniques to improve recommendation diversity," in *2013 6th International Conference on Biomedical Engineering and Informatics (BMEI 2013)*, 2013, pp. 661–665.
[4] S. Gong, H. Ye, and P. Su, "A peer-to-peer based distributed collaborative filtering architecture," in *2009 International Joint Conference on Artificial Intelligence*, 2009, pp. 305–307.
[5] T. Sandholm and H. Ung, "Real-time, location-aware collaborative filtering of web content," in *CaRR 2011*, 2011, pp. 14–18.
[6] C. Zeng, C.-X. Xing, and L.-Z. Zhou, "Similarity measure and instance selection for collaborative filtering," in *WWW 2003, May 20-24, 2003, Budapest, Hungary.*, 2003, pp. 652–658.
[7] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, "Item-based collaborative filtering recommendation algorithms," in *SIGIR*, 2001, pp. 285–295.
[8] Y. Kim and K. Shim, "Parallel top-k similarity join algorithms using Map-Reduce," in *2012 IEEE 28th International Conference on Data Engineering*, 2012, pp. 510–521.
[9] A. Tripathy, A. Patra, S. Mohan, and R. Mahapatra, "Designing a collaborative filtering recommender on the single chip cloud computer," in *2012 SC Companion:High Performance Computing,Networking Storage and Analysis*, 2012, pp. 838–847.
[10] C. V. Ramamoorthy and H. F. Li, "Pipeline architecture," in *Computiing Surveys*, 1977, pp. 62–102.
[11] B. N. Miller, I. Albert, S. K. Lam, J. A. Konstan, and J. Riedl, "MovieLens unplugged: Experiences with an occasionally connected recommender system," in *IUI*, 2003, pp. 263–266.
[12] V. Kumar and V. Singh, "Scalability of paralled algorithm for the all-pairs shortest path problem," in *The proceedings of the 1990 International Conference on Parallel Processing*, 1990, pp. 136–140.