Improving the Scalability of ALS-based Large Recommender Systems with Similar User Index

Haesung Lee* and Joonhee Kwon**

*,**Department of Computer Science, Kyonggi University,
San 94-6, Yiui-dong, Yeongtong-ku, Suwon-si, Gyeonggi-do, Korea
{seastar0202, kwnjh}@kgu.ac.kr

Abstract

Alternating Least Squares (ALS) is popular method to compute matrix factorization in the parallel way. However, due to the time complexity in predicting user’s preference, ALS is not scalable to large-scale datasets. In this paper, we propose a similar user index-based parallel matrix factorization approach. Since the group of similar users is indexed in advance, there is no need to compute similarities between all users in datasets. Furthermore, the size of a matrix is reduced because the matrix is only composed of indexed user’s ratings and items. The current advanced cloud computing including Hadoop, MapReduce and Amazon EC2 are employed to implement the proposed approaches. We empirically show that the use of similar user index resolves the scalable issue of ALS and improves the performance of large scale recommender systems in distributed computing environment.

Keywords: Scalable Recommender systems, Similar user index, ALS, Cloud computing

1. Introduction

Recommender systems suggest items that are likely to interest the users. Generally recommender systems are based on CF (Collaborative Filtering) which is a technique that automatically predicts the interest of an active user by collecting and analyzing user’s historical data. The underlying assumption of collaborative filtering is that the active user will prefer those items which the similar user prefer [1]. To handle web-scale datasets with millions of users and billions of ratings, scalability becomes an important issue in the domain of recommender systems. However, most recommender systems based on CF suffer from two limitations, data sparsity and scalability.

MF(Matrix Factorization) has become one of the leading techniques for the data sparsity which is main limitation of CF. Among many MF-based approaches, the alternating least squares (ASL) appear to be the most widely used method. ASL is inherently suitable for parallelization. But, due to the cubic time complexity in composing recommendations, ALS is not scalable to large-scale datasets.

Cloud computing provides software and hardware platforms to facilitate scalable and parallel computing. With the development of cloud computing technologies such as Hadoop, MapReduce and Amazon EC2, it becomes easy and convenient for users with little cloud computing knowledge to handle large scaled datasets on clouds quickly.

In this paper, we propose similar user index-based parallel matrix factorization algorithms. With the use of similar user index, it is possible to reduce computation time needed in composing recommendations. Also, the use of the similar user index reduces the size of a matrix based on which similarities of users are computed. That is, with the advantage of the similar user index, ALS-based recommender system is efficiently improved in the scalability. In addition, we implemented our approaches with the recent advanced cloud computing techniques including Hadoop,
MapReduce and Mahout in order to solve the problem of high computation complexity of large scaled datasets.

The remainder of this paper is organized as follows. Section 2 surveys related works. Section 3 discusses an ALS-based recommender system with the use of the similar user index. Section 4 presents the experimental study and results. Finally, Section 5 concludes this paper.

2. Background and Related Works

2.1 Recommender Systems

The information overload becomes a very serious issues. Personalized recommender systems can provide effective solutions to deal with information overload. Recommender systems can be thought of as information filters which present only relevant information or contents to the user.

Collaborative filtering (CF) is the most successful recommendation technology and is used in many of the most successful recommender systems [1]. Basically, a CF problem is modeled as regularized low-rank matrix approximation. However, CF has two fatal limitations which are respectively related with the data sparsity and scalability [2]. Especially the scalability problem is an important issue for recommender systems. The large number of users, items and other information bring challenges to recommender systems. How to design and implement scalable recommender systems is very important.

Matrix factorization approaches have been applied successfully for both rating-based and implicit feedback-based CF problems [3-5]. MF methods perform a so-called low-rank matrix approximation. There are two commonly used approaches to perform the approximate matrix factorization task, gradient descent and alternating least squares (ALS). ALS is effectively used for implicit feedback dataset. Besides, it is widely known that ALS is inherently suitable for parallelization [6].

2.2 ALS

Bell and Koren proposed the alternating least squares approach for CF [7]. The prediction formula of ALS for user \( u \) and item \( i \) is \( \hat{r}_{ui} = p_u^T q_i \). The parameters of the model \( p_1, \ldots, p_U \in \mathbb{R}^F \) and \( q_1, \ldots, q_I \in \mathbb{R}^F \) are called user and item feature vectors when \( F \) denotes the number of features. Let \( P \) and \( Q \) denote the matrices that contain user and item feature vectors as rows. The objective function associated with the model is \( \hat{f}^I \), thus the goal of ALS is to approximate the rating matrix as \( R \approx PQ^T \) so that the weighted squared error of the approximation is low. The key idea of ALS is that although the objective function consists of \( U \cdot I \) terms, its ALS based minimization can be done efficiently. The outline of the ALS is described in Algorithm 1.

<table>
<thead>
<tr>
<th>Algorithm 1. Alternating least squares (ALS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initialize ( Q ) with small random numbers</td>
</tr>
<tr>
<td>for ( E ) times do</td>
</tr>
<tr>
<td>\hspace{1cm} Compute the ( P ) that minimizes ( f_I ) for fixed ( Q ).</td>
</tr>
<tr>
<td>\hspace{1cm} Compute the ( Q ) that minimized ( f_I ) for fixed ( P ).</td>
</tr>
<tr>
<td>end</td>
</tr>
</tbody>
</table>

A nice property of ALS is that it does not replace objective function by an approximation, as other approaches often do. It achieves speedup by applying mathematical simplification. However, the more the matrix size increases, the more the
computation time of ALS is needed. That is, ALS is not efficient for dealing with the large-scale dataset.

2.3 Cloud Computing

Cloud computing doesn’t yet have standard definition, but a good working description of it is to say that clouds, or clusters of distributed computers [8]. Two different but related types of clouds are those that provide computing instances on demand and those that provide computing capacity on demand. Although both use similar machines, the first is designed to scale out by providing additional computing instances, whereas the second is designed to support data or computing intensive applications via scaling capacity. Amazon’s EC2 services are an example of the first type. And Hadoop systems and MapReduce are examples of second type.

Apache Hadoop is an open software framework that supports data-intensive distributed applications. It enables applications to work with thousands of nodes and petabytes of data [9]. Cascading is fault tolerant data processing workflows on a Hadoop cluster. As its simplicity and user friendly, cascading has been officially supported by Amazon Elastic MapReduce [10]. The Cascading flow will be converted into MapReduce jobs that can be executed on a Hadoop cluster with an inner MapReduce Job Planner [11].

MapReduce is a framework for processing huge datasets on certain kinds of distributed problems using a large number of computers or nodes, collectively referred to as a cluster [12]. Computational processing can occur on data stored either in a file system such as HDFS (Hadoop Distributed File Systems). The MapReduce framework consists of two parts, Map and Reduce. For the Map part, the master node takes the input, chops it up into smaller sub-problems, and distributes those to worker nodes. A worker node processes that smaller problem, and passes the output back to its master node. For the reduce part, the master node takes all outputs from each worker node and combines them in a way to get the final output.

3. ALS-based Recommender System with the Use of Similar User Index

3.1 Similar User Index

With the use of the similar user index, unnecessary data is efficiently filtered out and the matrix size of recommender algorithms is reduced. Therefore the recommender system with similar user index can quickly compose recommendation.

In order to index similar users in advance, we use the cluster technique. Similar user clustering techniques work by identifying groups of users who appear to have similar preferences. Major task of similar user clustering is measuring the distance between a randomly selected seed user and a specific user. If the distance between the user A and the seed of some cluster R is shorter than seeds of other clusters, the user A is included in the cluster R. The distance is considered as similarity of the user and the seed user. For measuring the similarity, we use the square of the Euclidean distance measurement [13].

Equation (1) is for measuring the distance between the seed user, $seed_i$ and the user, $u$ based on Euclidean distance measurement.
\[ \text{Sim}_{seed_i,u} = \frac{1}{\sqrt{(TN(\text{seed}_i) - MN(\text{seed}_i,u)) + (TN(u) - MN(\text{seed}_i,u))}} \] (1)

In equation (1), \( TN \) is total number of items experienced by \( seed_i \) and \( u \). \( MN \) is the number of items experienced by both \( seed_i \) and \( u \) in common. Algorithm 2 takes similar user clustering.

**Algorithm 2. Similar user clustering**

**Input:** \( k \) is the number of clusters, \( U \) is the user dataset  
**Output:** The set of clusters, \( C \)  
1. Randomly select and let \( s_1...s_k \) be the initial cluster centers.  
2. for each cluster \( c_k \in C \)  
3. do \( \text{centroid}_k \leftarrow \text{Position}(s_k) \)  
4. repeat  
5. for each user node \( u_i \in U \) do  
6. for each seed \( s_k \) do  
7. \( \text{similarity}_{i,k} \leftarrow \text{Euclidean}(u_i, \text{centroid}_k) \)  
8. \( \text{centroid}_k \leftarrow \text{ReadjustCentroid}(c_k) \)  
9. if \( \text{similarity}_{i,k} \) is biggest value among other clusters  
10. then \( c_k \leftarrow u_i \)  
11. until the centroid of each clusters do not change  
12. return \( C \)

In algorithm 2, \( \text{centroid}_k \) is center position of cluster \( c_k \). The function, \( \text{Position} \) returns the coordinate location in the cluster. We define initial \( \text{centroid}_k \) as the center position of each cluster. \( \text{Euclidean} \) measure the distance between the each user and the center position of each cluster. \( \text{ReadjustCentroid} \) readjust the center position in each cluster based on recently computed similarity. The cluster \( c_k \) which has the shortest distances between the user \( u_i \) and the center position is chosen to include the user \( u_i \). The clustering computation is iterated until the center position of each cluster doesn’t changed.

### 3.2 Large Scale Implementation

To solve the problem of high computation complexity of predicting user preferences and composing recommendations, we propose the use of similar user index. Since cloud computing including Hadoop and MapReduce can largely simplify the complexities with large-scale system development, job creation and job scheduling, we implemented the proposed approach with the cloud computing technique.

Figure 1 illustrate the data flow of generating the similar user index on the Hadoop system. In Figure 1, \( u_i \) is a user and \( r_j \) is an item. Through the task of ExtractData, the list of the user \( u_i \) and the item \( r_j \) which is experienced by \( u_i \) is generated. CoE checks whether two users \( u_i \) and \( u_{i-1} \) commonly experience the item \( r_j \). If the item \( r_j \) is experienced by both \( u_i \) and \( u_{i-1} \), 1 is assigned as an integer value. Count adds up all the
assigned integer value 1 and the total value is assigned as \(\text{summationValue}\). The similarity of \(u_i\) and \(u_{i-1}\) is calculated by the task of \(\text{Sim}\). With calculated similarities, \(\text{AssignCID}\) decides which the cluster user \(u_i\) is belonged. Finally, through the task of \(\text{GroupBy}\), the file of \(\text{similarUserIndex}\) is generated and saved in a file system.

**Figure 1. The Data Flow of Similar User indexing in the Hadoop system**

In order to implement similar user index with Hadoop framework, we define the MapReduce algorithm. Algorithm 3 takes the MapReduce processing for the generation of the similar user index.

**Algorithm 3. MapReduce of similar user indexing**

**procedure:** SimilarUserIndexer_Mapper_1  
**input:** history records  
**output:** \((\text{userTuple, <itemID,1>})\) pairs  
**begin**  
1. initiate temp_ketValue_vucket \(<\text{userID, item_list}>\)  
2. for each userID \(i\) in history records  
3. item_list ← all items experienced by \(i\)  
4. Add \((i, \text{item_list})\) in temp_ketValue_vucket  
5. for each userID \(i\) in temp_ketValue_vucket  
6. for each itemID \(k\) in item_list of userID \(i\)  
7. for each itemID \(l\) in item_list of userID \(j\)  
8. if \(k == j\)  
9. then emit \((\text{userTuple}(i,j), <k,1>)\)  
**end**

**procedure:** SimilarUserIndexer_Reducer_1  
**input:** \((\text{userTuple, <itemID,1>})\) pairs  
**output:** \((\text{userTuple, <match count>})\) pairs  
**begin**  
1. initiate temp_ketValue_vucket1 \((\text{userTuple, <itemID,1>})\) pairs  
2. initiate match_count  
3. for each userID \(i\) in temp_ketValue_vucket1  

Copyright © 2015 SERSC
4.   match_count = 0
5.   initiate temp_ketValue_vucket2
6.   for each userTuple j in temp_ketValue_vucket1
7.     if i == j
8.       then
9.         match_count ← match_count + 1
10. emit (i, match_count)
end

procedure: SimilarUserIndexer_Mapper_2
input: (userTuple, match_count)
output: (userTuple, distance) pairs
begin
1. initiate temp_ketValue_vucket (userTuple, match_count)
2. for each userTuple i in temp_ketValue_vucket
3.    distance ← Distance(k in i, l in i)
4.    emit (i, distance)
end

procedure: SearchRecommendedItem_Reducer_2
input: (userID, recommended_itemID_list) pairs
output: (userID, recommended_itemIDList) pairs
begin
1. initiate itemIDList
2. for each itemID i in recommended_itemID_list
3.   Add i in itemIDList
4. emit (userID, itemIDList)
end

AS being described in Algorithm 3, MapReduce of similar user indexing is consisted of two pairs of MapReduce functions. In other words, the output of the first MapReduce function is the input of the second MapReduce function.

4. Evaluation

We confirmed the advantage of the similar user index through the experiment in this section. For the experiment, we implement two recommender systems with the service of Amazon EC2 Elastic MapReduce. While both recommender systems are ALS-based, one doesn't use the similar user index and the other uses the similar user index in composing recommendations. In the comparison experiment, we call the similar user index-based recommender system as ALS+SimIndex, and the other is simply ALS.

4.1 Datasets

We conducted the experiment with the MovieLens dataset [14]. The MovieLens is an online movie recommendation site. The dataset contains 10 million ratings applied 10,000 movies by 72,000 users retrieved over various periods of time. The full corpus is about 10M of compressed data. It’s one of the largest datasets officially provided by MovieLens. The details of the dataset are discussed in [15]. Based on the largest dataset from MovieLens, we composed 10 data sets with increasing data size. Figure 2 shows synthetically composed 5 data sets.
Figure 2. Synthetically Composed 5 Data Sets with Increasing Data Sizes

4.2 Experimental Environment

The experimental environment includes the development environment and the computation environment. The development environment is locally constructed with 2.4 GHZ Intel Core i5 and 8GB of 1600MHZ DDR3 RAM. The operating system is Ubuntu 12.04 and Java is used as programming language and JDK 1.7 was installed. Eclipse is used as the programming and building tools. Hadoop 2.2 is installed and the stand alone mode of Hadoop was used to debug the programs. We use the library of Apache mahout 0.8 in order to efficiently implement ALS-based recommender system. Apache mahout is the machine learning library and provides useful libraries for recommender systems [16].

The computation environment is Amazon EC2 Elastic MapReduce (Amazon EMR) clouds shown like Figure 3.
run in the AWS console mode, which makes it easy and convenient to submit and run the customer applications or jobs.

### Table 1. The Service Type of Amazon EMR

<table>
<thead>
<tr>
<th>Instance Name</th>
<th>Memory (GB)</th>
<th>EC2 Compute Units and Cores</th>
<th>Platform</th>
<th>I/O Performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extra-large</td>
<td>15</td>
<td>8 EC2</td>
<td>64-bit</td>
<td>High</td>
</tr>
</tbody>
</table>

Since the AWS console supports the job submission and the management of jobs with web browser, it becomes easy to submit and run custom jobs in Amazon EC2 [17]. In Elastic MapReduce of AWS Management Console, we can create a new workflow and select customer jar as the job type. Table 1 shows the service type of Amazon EMR used in our computation environment. We select the service type, Extra-large which is enough for our experimental environment. With 8EC2, we can increase the number of cores from 1 to 8 for evaluating the scalability of recommender systems.

### 4.3 Results and Discussions

To evaluate the effectiveness of the proposed similar user index in improving the scalability of recommender systems, we created eight running jobs on Amazon EC2 EMR clouds. And we conducted two comparison experiments with $ALS+SimIndex$ and $ALS$ respectively on EC2 clouds. Eight running jobs are different with the number of cores.

![Figure 4. The Scalability Comparisons of Two Recommender Systems with Two Data Sets, SD_1 and SD_2](image)

![Figure 5. The Scalability Comparisons of Two Recommender Systems with Two Data Sets, SD_3 and SD_4](image)
In both comparison experiments, we measure each running time that is consumed during the composition of recommendations on 8 cores respectively. And then, we investigate that how quicker each running time is with increasing cores. We define the degree of the quicker running time as speedup.

Figure 4 comparatively shows the experimental results between two recommender systems. Figure 4 (a) and Figure 4 (b) show experimental results for dataset SD_1 and SD_2 respectively. And Figure 5 (a) and Figure 5 (b) show experimental results for dataset SD_3 and SD_4 respectively. Also, Figure 6 shows the scalability comparisons for two recommender systems with the data set, SD_5. With all experimental results, while both recommender systems ALS+SimIndex and ALS are quicker in composing recommendations with increasing cores, speedup of ALS+SimIndex is bigger than ALS. In other words, ALS+SimIndex more quickly reduces the running time with increasing cores than ALS which doesn’t use the similar user index. The more the number of cores increase, the more the size of computed data on a core is reduced. In addition, when recommender algorithms use the similar user index for predicting user’s preference, the size of computing data is reduced. Therefore, in the case of ALS+SimIndex, the size of computing data on each core is smaller than that in ALS.

As a result, the recommender system with the use of similar user index is superior to the recommender system in which only ALS method is applied without the use of similar user index. So the use of similar user index efficiently improve the scalability of recommender systems.

5. Conclusion

The scalability issue has been recognized as one of the crucial challenges that most recommender systems confront. ALS is inherently suitable for parallelization. However, due to the time complexity in composing recommendations, ALS-based approaches are inefficient in dealing with a huge amount of historical data.

In this paper, we propose a new approach that ALS-based systems use the similar user index in composing recommendations. With the use of the similar user index, unnecessary data is efficiently filtered out and the matrix size of recommender algorithms is reduced. Therefore the recommender system with similar user index can quickly compose recommendation. Through the experimental result, we verify that the recommender system with the use of similar user index is superior to the recommender system in which only ALS method is applied without the use of similar user index.

In conclusion, the use of proposed approach improves the scalability of the ALS-based recommender systems. So, the similar user index is usefully applied in the large-scale recommender systems which handle a huge amount of data.
References


[7]. R. M. Bell and Y. Koren. Scalable collaborative filtering with jointly derived neighborhood interpolation weights. In ICDM-07, 7th IEEE Int. Conf. on Data Mining, pages, Omaha, Nebraska, USA, 43–52 (2007)


Authors

Lee Haesung is a Ph.D. candidate of Computer Science at Kyonggi University, Korea. Her research areas include Context-aware Computing, Social Network, Information Retrieval and Mobile Computing. She works for software development on areas of data search in ubiquitous environment. She received her B.S., M.S. in Computer Science from Kyonggi University, Korea. Contact her at seastar0202@kyonggi.ac.kr.

Kwon Joohhee is a professor of Computer Science at Kyonggi University, Korea. She was a visiting research professor at the Computer Science Department at New Jersey Institute of Technology. Her research areas include Context-aware Computing, Information Retrieval, Social Network, Web 2.0 and Mobile Database. Her research projects focus on areas of data search using social network and Web 2.0 in ubiquitous environment. She received her B.S., M.S. and Ph.D. in Computer Science.
Science from Sookmyung Women’s University, Korea. Contact her at kwonjh@kyonggi.ac.kr.

*Corresponding author: Joonhee Kwon, Professor. Department of Computer Science, Kyonggi University, San 94-6, Yui-dong, Yeongtong-ku, Suwon-si, Gyeonggi-do, Korea
E-mail: kwonjh@kgu.ac.kr