Scalable Recommender System over MapReduce

Zhong Wang, Shuyang Yao
Department of Computer Science, Duke University
LSRC Building D101, 308 Research Drive
Durham, NC 27708
{zw, shyyao}@cs.duke.edu

Abstract

In this paper, we use item-based and user-based collaborative filtering recommendation algorithms to predict user’s rating on the Movielens[3] dataset. We use RMSE as our metric to evaluating the performance of these two algorithms with different parameters. Besides, we implemented the scalable item-based recommender system over MapReduce. By comparing with the in-memory algorithm, we can find that the parallelized implementation is much more efficient on large datasets.

1 Introduction

Collaborative filtering(CF) is the most widely used recommendation algorithm these days, CF is a very classic method, it not only has satisfactory performance, but is also pretty efficient. Although it has been published for years, few algorithms can outperform it.

However, for large global business vendors with millions of users generating hundreds of millions of ratings, the feasibility of collaborative filtering will be restricted by the limited computing ability of a single machine[5]. In the forth section, we will discuss distributed collaborative filtering algorithms. We implement item-based algorithm over MapReduce to make recommendation on the MovieLens 10M[7] dataset, which contains more than 10 million ratings from 71,567 users rating on 10,681 movies.

2 Problem Description

In the problem of recommendation, we have a list of users \(U = \{user_1, user_2, ..., user_n\}\), and also a number of items \(I = \{item_1, item_2, ..., item_m\}\), each user have some ratings for a small subset of items \(rating_i = \{r_{i1}, r_{i2}, ..., r_{ik}\}\), we need to predict \(user_i\)'s rating for other unrated items based on \(rating_i\) and other similar users’ rating. According to the predicted rating, we can then recommend new items to \(user_i\). In other word, we already have a rating matrix in hand, each row represents a user’s rating to all the items, and each column represents all users’ rating toward one item, our goal is to fill in all the empty cell in this matrix based on the initial matrix.

3 Related Works

Recommender System is a topic that has been well studied and applied. Among different kinds of approaches for recommending, collaborative filtering is the one that has the best balance between efficiency and accuracy. Within the family of collaborative filtering, Item-based collaborative filtering is the most widely used algorithms, which was proposed by GroupLens Reasearch Group and Army HPC research Center in 2001[9]. There are also other type of recommender algorithms, for instance,
Sarwar used singular value decomposition (SVD) in recommender systems.[8] Breese proposed two model-based collaborative filtering algorithms in [3].

An engineering practice of building a distributed recommender system is using the pseudo-distributed collaborative filtering model [2]. The basic idea of this model is to build the whole recommender on each of the $p$ machines, but distribute the user recommendation process.

Other parallel collaborative filtering algorithms are based on matrix calculation, such as Alternating-Least-Squares with Weighted-$\lambda$-Regularization (ALS-WR) [11]. The algorithm iteratively minimizes the mean-square loss function with an empirical Tikhonov regularization term.

However, these methods are not effective enough due to either the single-machine limitation or low cost-effectiveness.

4 Collaborative filtering

We evaluate both item-based and user-based collaborative filtering on single machine in our project. We use Pearson correlation as similarity function. Item-based and user-based algorithms are thoroughly described in [10], user-based prediction is denoted in the following equation.

$$r_{u,i} = \frac{\sum_{u' \in U}(u', i)}{\sum_{u' \in U} |simil(u, u')|}$$

$U$ is the user’s neighbor set. Item-based prediction’s equation is similar.

5 Distributed Collaborative Filtering

5.1 Item-based Collaborative Filtering based on MapReduce

In order to parallelize the process of building the recommender with high scalability, we present an implementation of item-based collaborative filtering based on MapReduce.

5.1.1 MapReduce Model

MapReduce[4] is a programming model, which is widely used in large-scale data processing and parallel computing. Basically, there are two functions in MapReduce model:

$$map(k_1, v_1) \rightarrow list(k_2, v_2)$$

$$reduce(k_2, list(v_2)) \rightarrow list(v_2)$$

The underlying idea of MapReduce for parallel computing is that, we can distribute the computation into independent $map$ functions, then aggregate the result from the distributed $map$ functions into $reduce$ functions, which can also be distributed, while ensuring that each $reduce$ function processes all values of $k_2$ from different $map$ functions.

5.1.2 User-based or Item-based

There are two major reasons that item-based collaborative filtering outperforms user-based method when datasets getting larger: 1) item-based method has better performance when the rating data is sparse for users, which is a common case on large datasets [9]; 2) the cost for calculating $m^2$ pairs of item similarities is much smaller than calculating $n^2$ pairs of user similarities.

5.1.3 Implementation

We implement the distributed item-based collaborative filtering in 3 MapReduce phases:

**Phase 1.** Preprocessing: transform each line of the raw data into the form of 
\{userId, list(itemId, rating)\} to prepare data for the next phases:

$$map(line) \rightarrow list(k: userId, v: (itemId, rating))$$
reduce(k : userId, v : list(itemId, rating)) →

k : userId, v : list(itemId, rating)

Phase 2. Calculating similarity: take the output of Phase 1 as input. The map function emits all pairs of movies with ratings that the user rated. The reduce phase calculate the similarity (e. g. Pearson) of a pair of movies, and outputs the item-item similarity:

map(k : userId, v : list(itemId, rating)) →

list(k : (itemId1, itemId2), v : (rating1, rating2))
reduce(k : (itemId1, itemId2), v : list(rating1, rating2)) →

k : (itemId1, itemId2), v : similarity

Phase 3. Recommendation: in this phase, we calculate the ratings of unrated items based on the item similarity matrix. We have two map functions in this phase. The first map function takes the output of Phase 1, and builds an inverted index from itemId to a list of userId. The second map function takes the item similarity matrix as input, and emits the neighbors and the corresponding similarities of each item. The reduce function calculate the ratings for unrated items based on the similarities of neighbors.

map1(k : userId, v : list(itemId, rating)) →

list(k : itemId, v : (userId, rating))
map2(k : (itemId1, itemId2), v : similarity) →

k : itemId1, v : (itemId2, similarity)
reduce → k : userId, v : list(itemId)

Several optimizations could be applied to this implementation. For example, we could set a threshold for the least similarity to control the output size of Phase 2. Or, we could add another phase to calculate top-n nearest neighbors of items.

6 Experiment

6.1 user-based and item-based CF on single machine

We evaluate the performance of item-based and user-based collaborative filtering algorithm on the MovieLens 1-m dataset by using 10 fold cross validation on a single machine. We use RMSE(Root Mean Square Error) as our criteria. We test how the size of the neighbor set affects the algorithms' performance.

Figure 1: result comparison between item-based and user-based CF
Figure 1 illustrates the comparison between user-based and item-based algorithm, the curve in the figure is the average RMSE among 10-fold cross validation.

One reason why item-based algorithm is widely used is that for most E-Commerce sites the number of item is much smaller than the number of users, so it is not only much more efficient to use item-based method, but also more accurate, because in these cases, ratings are more sparse in term of users than items. However, it is not always the case, in a news site, the number of articles are obviously much more than the number of users. Using item-based article will be inefficient.

We also find that the items which each algorithms recommend to a user is very different. From Figure 2 we can see a very low overlapping rate between two algorithms’ result, only half of the recommended items are same when both algorithms recommend 1500 items to a user. With regard that two algorithms have similar accuracy, we can say that their recommendations are complementary.

6.2 Item-based over MapReduce

In order to test the scalability of the distributed algorithm, we applied item-based recommendation on MovieLens 10M dataset based on Apache Hadoop 1.2.1[1]. The experiments ran on 12 Linux boxes, each with $2 \times 4$ Xeon CPU cores (L5420 - 2.50GHz), $2 \times 4$GB DDR2 RAMs and $1 \times 1$TB SCSI disk. Each Hadoop TaskTracker is configured with 16 map slots and 4 reduce slots.

In order to better demonstrate different models, the whole recommendation process is separated into two phases: item similarity and recommendation. For item similarity phase, the item-item similarities are calculated in $O(N^2)$ time, where $N$ is the number of items (10,681 in MovieLens 10M dataset). T

We compared the running time of MapReduce implementation with single-machine in-memory implementation. As shown in Figure 3, the growth of running time of mapred is slower than in-mem, which implies better scalability – when dataset is getting larger, the mapred implementation will have better performance. However, on the limited size of MovieLens 10M dataset, the running time of item similarity of mapred is still longer than in-mem. This is because of the high overhead for bootstrap, network and disk I/O on MapReduce framework.

Figure 4 demonstrates the effectiveness of MapReduce implementation in the recommendation phase. As can be seen, along with the increase of the number of users to be recommended, the in-mem implementation leads a linear trend of running time, while mapred remains a constant and much lower running time.

The main difference between the two phases is that, the item similarity phase generates an $O(N^2)$ similarity matrix, which leads to large amount of I/O consumption on MapReduce framework. By contrast, the recommendation phase does not generate such large intermediate data, and not fully
Figure 3: Running time of item similarity phase. The MovieLens 10M dataset is sampled from 1M ratings to 10M ratings to evaluate the growth rate of each method.

Figure 4: Running time of recommendation phase. We measure the effectiveness of the two methods by increasing the number of users to be recommended.

utilize the computation capacity of all the nodes. Thus, the running time remains a constant level due to the power of parallelism.

7 Conclusion

From our experiment, we can see that although item-based and user-based algorithms have similar accuracy, they are recommending totally different items. Item-based algorithm recommend items that are similar to a user’s current preference, so the recommender system is not likely to give innovative recommendation, while user-based algorithm is able to recommend totally new items to a user. However, User-based algorithm has “cold-start” problem, while Item-based algorithm can avoid the problem.

Besides, we implement item-based recommendation over MapReduce framework. The result shows that, the parallelized implementation is much more efficient than single-machine version on large datasets. Along with the coming of the Big Data era, the scalable recommender system will play a more important role in varieties of applications for modern business.
References
