User-based Collaborative-Filtering Recommendation Algorithms on Hadoop

Zhi-Dan Zhao
School of Computer Science and Engineering
University of Electronic Science and Technology of China
610054 Chengdu, P. R. China
yashiki@126.com

Ming-Sheng Shang
School of Computer Science and Engineering
University of Electronic Science and Technology of China
610054 Chengdu, P. R. China
msshang@uestc.edu.cn

***Abstract*—Collaborative Filtering(CF) algorithms are widely used in a lot of recommender systems, however, the computational complexity of CF is high thus hinder their use in large scale systems. In this paper, we implement user-based CF algorithm on a cloud computing platform, namely Hadoop, to solve the scalability problem of CF. Experimental results show that a simple method that partition users into groups according to two basic principles,i.e., tidy arrangement of mapper number to overcome the initiation of mapper and partition task equally such that all processors finish task at the same time, can achieve linear speedup.**

***Keywords*-collaborative filtering; recommender systems; cloud computing; hadoop; Map-Reduce;**

I. Introduction

Collaborative Filtering(CF) algorithm is a widely used personalized recommendation technique in commercial recommendation systems[1], [2], and many works have been down in this field to improve the performance[3], [4]. However, a big problem of CF is its scalability, i.e.,when the volume of the dataset is very large, the computation cost of CF would be very high. Recently, cloud computing have been the focus to overcome the problem of large scale computation task. Cloud computing is the provision of dynamically scalable and often virtualised resources as a service over the Internet[5]. Users need not have knowledge of, expertise in, or control over the technology infrastructure in the ”cloud” that supports them. Cloud computing services often provide common business applications online that are accessed from a web browser, while the software and data are stored on the servers.

In order to solve scalability problem of recommender system, we implement the Collaborative Filtering algorithm on the cloud-computing platform. There are several cloud-computing platforms available, for example, the Dryad [6] of Microsoft, the Dynamo [7] of *amazon.com* and Nettune [8] of *Ask.com* etc. In this paper, we choose the Hadoop platform as the base of our implement. Because the Hadoop platform [9], [10]is an open source cloud computing platform, it implements the MapReduce framework that have been successfully evaluated by *Google.com*. The Hadoop platform uses a distributed file system, Hadoop Distributed File System(HDFS)[11], to provide high throughput access to application data. Using the Hadoop platform, we can easily make the program execute in parallel, and the MapReduce framework allows the user to break a big problem for many small problems, then the small problems could be handled by the Hadoop platform, thus improve the speed of computing.

As for the implement of Collaborative Filtering algorithm on cloud computing platform, there are few work have been done. Abhinandan Das and Mayur Datar[12] proposed a Collaborative Filtering algorithm for news recommendation, which is a combining of memory based and model based Collaborative Filtering algorithm. There is also a very close rated project, *Mahout* [14] , which implement the Collaborative Filtering recommendation system base on *Taste* [15],a flexible, fast collaborative filtering engine for Java.

In this paper, we study the implement of collaborative filtering algorithm on cloud computing platform. The work we have done are summarized as follows. Firstly, We designed a user based collaborative filtering algorithm for the MapReduce program famework, and implement the algorithm on the Hadoop platform. Secondly, we tested our implement under several configurations and found that a simple method that partition users into groups according to two basic principles,i.e., tidy arrangement of mapper number to overcome the initiation of mapper and partition task equally such that all processors finish task at the same time, can achieve linear speedup.

The rest of this paper is organized as follows: Section 2 discusses the Collaborative Filtering process and the MapReduce programming model. Section 3 presents the implementation of the Collaborative Filtering algorithm on hadoop platform. Section 4 shows our experiment results and our analysis. Section 5 concludes the paper with pointers to future work.

II. CF And MapReduce

A. Collaborative Filtering

Collaborative Filtering algorithm is a classic personalized recommendation algorithm, it’s widely used in many commercial recommender systems[1]. Collaborative Filtering algorithm is an algorithm based on the following three

978-0-7695-3923-2/10 $26.00 © 2010 IEEE
DOI 10.1109/WKDD.2010.54

assumptions idea: People have similar preferences and interests; Their preferences and interests are stable; We can predict their choice according to their past preferences. Because of the above assumptions, the collaborative filtering algorithm is based on the comparison of one user's behavior with other user's behavior, to find his nearest neighbors, and according to his neighbor's interests or preferences to predict his interests or preferences.

The first step of collaborative filtering algorithm is to obtain the users history profile, which can be represented as a ratings matrix with each entry the rate of a user given to an item[1]. A ratings matrix consists of a table where each row represents a user, each column represents a specific movie, and the number at the intersection of a row and a column represents the user's rating value. The absence of a rating score at this intersection indicates that user has not yet rated the item. Owing to the existence problem of sparse scoring, we use the list to replace the matrix.

The second step is to calculate the similarity between users and find their nearest neighbors. There are many similarity measure methods. The Pearson correlation coefficient is the most widely used and served as a benchmark for CF. Generally we use the Cosine similarity measure method, it's calculate equation as follows:

$$sim(x,y) = \frac{\sum_{s \in S_{xy}} r_{x,s} r_{y,s}}{\sqrt{\sum_{s \in S_{xy}} r_{x,s}^2} \sqrt{\sum_{s \in S_{xy}} r_{y,s}^2}} \qquad (1)$$

Where r_x is rating of user x on item s and r_y is rating of user y on item s, S_{xy} indicates the items that user x and y co-evaluated.

The last step is to calculate the items rating. The rating is computed by a weighted average of the ratings by the neighbors[1].

$$r_{x,s} = \overline{r}_x + \frac{\sum_{y \in S_{xy}} (r_{y,s} - \overline{r}_x) sim(x,y)}{\sum_{y \in S_{xy}} sim(x,y)} \qquad (2)$$

$\overline{r}_x$ is the average rating of user x.

From the above steps we can find that the calculation process of Collaborative Filtering algorithm would consume intensive computing time and computer resources. When the data set is very large, the calculation process would continue for several hours or even longer. Therefore, we propose new method that is implement the Collaborative Filtering algorithm on Hadoop platform to solve that problem.

B. MapReduce Overview

The MapReduce model is a distributed implementation model which is proposed by Google com. Here, we introduce the MapReduce model and describe its working mechanism on Hadoop platform.

The MapReduce model is inspired by the Lisp programming language map and reduce operations. The MapReduce model abstract the calculation process into two phase: map phase and reduce phase. In the Map, written by the user, takes a set of input key/value pairs, and produces a set of output key/value pairs. The MapReduce library groups together all intermediate values associated with the same intermediate key *I* and passes them to the Reduce phase. In the Reduce, the function also written by the user, accepts an intermediate key *I* and a set of values for that key. It merges together these values to form a possibly smaller set of values. Typically, none output value or only one is produced per Reduce invocation.

In this paragraph we briefly analyze the scheduling mechanism of the Hadoop. In the Hadoop platform, at default the input data set size of one mapper is less than 64MB (or a file, but the file size is less than 64MB), when the file is larger than 64MB, the platform would split it into a number of small files which size less than 64MB automatically. For one input file, the Hadoop platform initialize a mapper to deal with it, the file's line number as the key and the content of that line as the value. In the map stage, the user-defined process deal with the input key/value and pass the intermediate key/value to the reduce phase, so the reduce phase would implement them. When the files block are computed completely, The Hadoop platform would kill the corresponding mapper, if the documents are not finish, the platform would chooses one file and initializes a new mapper to deal with it. The Hadoop platform should be circulate the above process until the map task is completed.

III. MapReduce for CF

In this section we present how to implement the Collaborative-Filtering algorithm within the MapReduce framework. According to the introduction of the second section we know that the calculation process of Collaborating Filtering algorithm is not easy to directly use the MapReduce model decomposition. So we come up the idea that compute the recommendation process for each user, the recommendation process is encapsulated in the Map function. That is to say, when we make recommendation, we would store the user ID which need to calculate in some txt files, then these files as the input of the Map function. The MapReduce framework initializes some mapper to deal with these user ID files. Our algorithm could be divided into the following 3 phase, Three phases relationship as following Figure 1:

A. Data partitioning phase

In the Data partitioning phase, we separate the user ID into different files, in these files, each row store a user ID. these files as the input files of map phase, the data partitioning should satisfy 2 principles as follows:

1 In the total running time, the proportion of computing time, the bigger the better. That means the most proportion of run time should be spent in the computation process, rather than frequently initialize the mapper. For

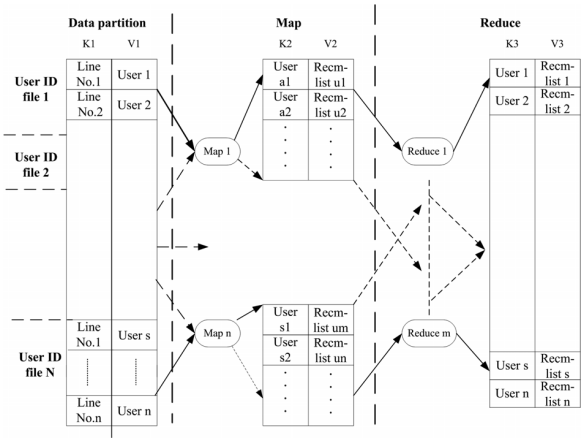

Figure 1. CF's MapReduce

example, we divide the 1000 users into 1000 copies, so that the Hadoop platform have to frequently initialize the mapper to deal with each copies. As a contrast, we divided the 1000 users into 40 copies and 50 copies. In this case, the platform would not frequently initialize the mapper.

2 The same end of the tasks running time. That is the end of each mapper task time should be at the same time. For example, when there are 10 mappers, we assume that other 9 mappers complete their tasks at time t, the last mapper complete its task at time t+100, if we distribute these 100 seconds on 10 mapper, only need 10 seconds to complete the task, that means at time t+10 all tasks are completed, but now, the Hadoop platform to run more than for 90 seconds.

B. Map phase

At this stage, the Hadoop platform estimate the algorithm's memory and others resources consumption, specified each DataNode the number of mapper it can be initialized. The Hadoop platform determines whether initialize a mapper to deal with the user ID files. If there have enough resources to initialize a mapper, the Hadoop platform initializes a new mapper. The mapper's setup function build the ratings matrix between the users and the items at first, the mapper read the user ID file by line number, take the line number as the input key and this line corresponding user ID as the value. The next step is to calculate the similarity between this user and other users. Equation 1 describes the similarity calculation process. The final step is to identify the user's nearest neighbors(by similarity values), and in accordance with the Equation 2 to calculate his predict rating on items. we sort the predict ratings and store them in recommend-list. The user ID and its corresponding recommend-list as the intermediate key/value, output them to the reduce phase. If the Hadoop platform without enough resources to initializes a new mapper, the platform has to wait for a mapper finish its task and release its resources, and then initializes a new mapper to deal with the user ID file. The process will continue until all tasks are completed.

C. Reduce phase

In the reduce phase, the Hadoop platform would generates some reducers automatically. The reducer collect the users ID and its corresponding recommend-list, sort them according to user ID, and then output them to the HDFS.

IV. Experiment results and Analysis

In this section we show our experiments results. We implement Collaborative Filtering algorithm on the Eclipse platform, we describe the CF'S process at section 3. Our Hadoop compute-cluster was composed by 9 computers, one computer as NameNode and other 8 computers as DataNodes. Each computer's memory is 4 GB and INTEL Dual-core 2.6 GHZ CPU, operating system Linux. In the experiments we use the Netflix data set, The Netflix data set contains more than 2.64 million movies and more than 17,000 users, the ratings of each user is not the same, the range is from several to tens of thousands of movies. However, as the article mainly compares the run-time between stand-alone and Hadoop platform, so that we don't consider the accuracy and recall. we take out 5 copies of sub-dataset as our experimental data sets, and the five copies of the data sets include 100 users, 200 users, 500 users and 1000 users. In the experiment, DataNode number is divided into 4 kinds, the four kinds are 2 nodes, 4 nodes, 6 nodes and 8 nodes. In order to compare the performance between stand-alone and Hadoop platform, when DataNode number and data set is fixed, we assign the user ID into different files, the Hadoop platform deal with these files, in accordance with the CF algorithm process which is described at section 3. at last, we take the average time T_a as the Hadoop platform at current DataNodes and data set running time. We define T_s as the stand-alone's running time. Speedup as an important criterion to measure our algorithm's superiority, we define the speedup as follows:

$$Speedup = \frac{T_a}{T_s} \tag{3}$$

At section 3 we describe the calculation process of Collaborative Filtering algorithm on Hadoop platform. As the recommendation process is based on the division of each user, when we use the Hadoop platform to make recommendations for many users, it is equivalent to assign the calculation on **N** nodes, so that in theory the Speedup should be **N**, **N** is the number of DataNode. In the ideal case, the Speedup should be linearly related to the number of DataNode. From the Figure 2 we can see that with the increase in the number of DataNode, the Speedup increase linearly. We also find that when there are 100 users and 200 users, the Speedup is not linear increase, this is because the Data set is too small, so the Hadoop platform has not demonstrates its superiority.

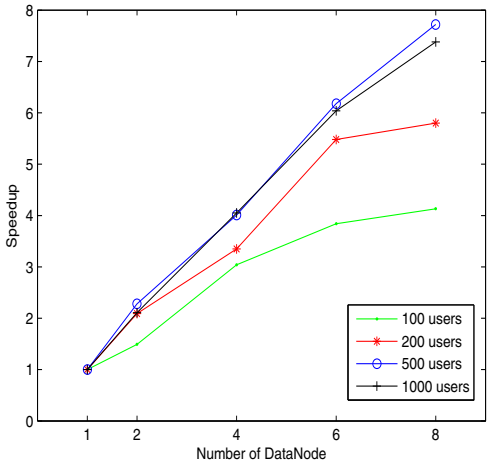

Figure 2. Speedup of CF

V. Conclusion

In this paper, we mainly addresse the challenge of using the MapReduce model to parallelize Collaborative Filtering. As a stateless programming model, MapReduce cannot directly express Collaborative Filtering. To achieve our goal, we have divided the calculation process by user ID, calculate the recommendation process for each user. The recommendation processes of the user is encapsulated in the Map function. The experiment result also shows that our design algorithms enable Collaborative Filtering algorithm in Hadoop platform to take the good performance.

Through the experiments we find that the main drawback of the MapReduce framework is that in the calculation process, whenever a new input file(or file blocks), it needs to initialize a mapper, and this process for some algorithms are very resource-consuming. The Collaborative Filtering algorithms on Hadoop platform can not to reduce the recommendation response time for a single user.

In our future works, we are planning to improve the MapReduce process, to enable the Hadoop platform to better deal with these algorithms with large computational but less input data.

Acknowledgment

This work was partially supported by the National Natural Science Foundation of China under Grant Nos. (60973069, 90924011), and by the China Postdoctoral Science Foundation under Grant No. 20080431273.

References

[1] Adomavicius G., Tuzhilin A., Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Trans. on Knowledge and Data Engineering*, 2005, 17(6): 734-749

[2] J. Herlocker, J. Konstan, L. Terveen, and J. Riedl. Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems*, 22(1):5-53.

[3] Shang Ming-Sheng, Zhang Zi-ke. Diffusion-Based Recommendation in Collaborative Tagging Systems.*Chin. Phys. Lett.*,2009,26(11): 118903

[4] Shang Ming-Sheng, Jin Ci-Hang, Zhou Tao, Zhang Yi-Cheng. Collaborative filtering based on multi-channel diffusion. *Physica A: Statistical Mechanics and its Applications*. 2009,388(23):4867-4871. M. D. Dikaiakos, D. Katsaros, G. Pallis, A.

[5] Vakali, P. Mehra: Guest Editors. *Introduction: "Cloud Computing, IEEE Internet Computing"*, 12(5), Sep. 2009.

[6] Isard M, Budiu M, Yu Y, Birrell A, Fetterly D. Dryad: Distributed data-parallel programs from sequential building blocks. *In: Proc. of the 2nd European Conf. on Computer Systems (EuroSys).*, 2007. 59 72.

[7] DeCandia G, Hastorun D, Jampani M, Kakulapati G, Lakshman A, Pilchin A, Sivasubramanian S, Vosshall P, Vogels W. Dynamo: Amazons highly available key-value store. *In: Proc. of the 21st ACM Symp. on Operating Systems Principles*. New York: ACM Press, 2007. 205 220.

[8] Chu LK, Tang H, Yang T, Shen K. Optimizing data aggregation for cluster-based Internet services. *In: Proc. of the ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming*. New York: ACM Press, 2003. 119 130.

[9] Dean J, Ghemawat S. Distributed programming with Mapreduce. *In: Oram A, Wilson G, eds. Beautiful Code*. Sebastopol: OReilly Media, Inc., 2007. 371 384.

[10] Dean J, Ghemawat S. MapReduce: Simplified data processing on large clusters. *Communications of the ACM*, 2005,51(1):107 113.

[11] Ghemawat S, Gobioff H, Leung ST. The Google file system. *In: Proc. of the 19th ACM Symp. on Operating Systems Principles*. New York: ACM Press, 2003. 29 43.

[12] Das A. S. , Datar M , Garg A and RajaramS. Google news personalization: scalable online collaborative filtering. *WWW '07: Proceedings of the 16th international conference on World Wide Web*,2007Banff, Alberta, Canadapages271-280

[13] Ranger C, Raghuraman R, Penmetsa A, Bradski G,Kozyrakis C. Evaluating MapReduce for Multi-core and Multiprocessor Systems. *High Performance Computer Architecture, 2007. HPCA 2007. IEEE 13th International Symposium on*,10-14 Feb. 2007, pages:13-24

[14] http://lucene.apache.org/mahout/

[15] http://taste.sourceforge.net/

[16] Vrba Z., Halvorsen P., Griwodz C., Beskow P. Kahn Process Networks are a Flexible Alternative to MapReduce", *High Performance Computing and Communications, 2009. HPCC '09. 11th IEEE International Conference on* ,25-27 June 2009, pages:154 - 162