Scaling-up Item-based Collaborative Filtering Recommendation Algorithm based on Hadoop

Jing Jiang, Jie Lu, Guangquan Zhang, Guodong Long

Decision Systems & E-Service Intelligence Research Laboratory
Center for Quantum Computing and Intelligent System, School of Software
Faculty of Engineering and Information Technology, University of Technology Sydney, Sydney,
POBOX123, Broadway, NSW2007, Australia

jing.jiang-1@student.uts.edu.au, {jie.lu, guangquan.zhang, guodong.long}@uts.edu.au

***Abstract*—Collaborative filtering (CF) techniques have achieved widespread success in E-commerce nowadays. The tremendous growth of the number of customers and products in recent years poses some key challenges for recommender systems in which high quality recommendations are required and more recommendations per second for millions of customers and products need to be performed. Thus, the improvement of scalability and efficiency of collaborative filtering (CF) algorithms become increasingly important and difficult. In this paper, we developed and implemented a scaling-up item-based collaborative filtering algorithm on MapReduce, by splitting the three most costly computations in the proposed algorithm into four Map-Reduce phases, each of which can be independently executed on different nodes in parallel. We also proposed efficient partition strategies not only to enable the parallel computation in each Map-Reduce phase but also to maximize data locality to minimize the communication cost. Experimental results effectively showed the good performance in scalability and efficiency of the item-based CF algorithm on a Hadoop cluster.**

Keywords- Collaborative Filtering; Cloud Computing; Hadoop; Mapreduce; Scalability; Parallelization

I. INTRODUCTION

Cloud computing, from the inception of its concept, has become popular in both industry and academia. The concept of cloud computing is similar to that of distributed computing, but the most difference is that cloud computing uses MapReduce as its programming model. MapReduce, first proposed by Google [1], is attracting a lot of attention, providing scalable computing power to process mass information over clusters of computers [2-4].

Hadoop [5], an popular open source cloud computing platform, uses MapReduce as its computing framework. It is very suitable for processing large-scale datasets, especially in the internet. It is implemented by Yahoo, where it processes hundreds of terabytes of data on thousands of common machine cluster [6]. In addition, it focus on providing the necessary minimum functionality and combining simplicity of use with scalable performance, rather than copying with other complicated and confused procedures, such as fault tolerance, inter-communication, resource allocation and synchronization, etc. [7].

The exponential increase in sources of information available in the World Wide Web has created a challenge to find ways to identify relevant information and knowledge. Intelligent approaches are needed to provide users to efficiently locate and retrieve information. Thus, recommendation systems have emerged to assist business and have increasingly become more important solutions to deal with the information overload problem. One of earliest used and most successful recommendation techniques is collaborative filtering (CF) [8-14]. They are widely used in many commercial recommendation systems in past [15]. However, their widespread use has revealed some potential challenges, such as scalability [16].

In a CF algorithm, it is expensive to compute the similarity of users as the algorithm is required to search entire database to find the potential neighbors for a target user. CF algorithms require computation that increase linearly with the growth of both the number of users and the number of items. Therefore, many algorithms are either slowed down or require additional resources such as computation power or memory. This is so-called the scalability problems [17]. To solve the scalability of CF algorithm, Zhao and Shang [8]implement CF algorithm on Hadoop. However, their method of dividing data sets has not favorable scalability and computation-cost efficiency if the datasize increase.

In this paper, we proposed a method by implement a scaling-up item-based CF algorithm on MapReduce in a Hapdoop cluster. We split the costly computation into many different small computing partitions, each of which can independently execute on different nodes in parallel. We also proposed effi-

978-0-7695-4461-8/11 $26.00 © 2011 IEEE
DOI 10.1109/SERVICES.2011.66

IEEE computer society

cient partition strategies to maximize data locality to reduce the communication cost, and to control the algorithm complexity to increase computation power, thereby to obtain good scalability even if the large-scale datasets.

The rest of the paper is structured as follows. In Section 2 we introduce the MapReduce framework and the main idea of our algorithm. In Section 3 we present partition strategies for Map-Reduce schemes. Next, the experimental evaluation is reported in Section 4. Finally, the contributions of this paper and future work are summarized in Section 5.

II. PRELIMINARIES

In this section, we introduce the MapReduce computation framework in Hadoop and the item-based collaborative filtering algorithm.

A. Overview of MapReduce

MapReduce [1, 18] is a popular programming model proposed by Google. MapReduce is ideally fit for data already stored on a distributed file system which offers data replication as well as the ability to perform computation locally on each data node[3, 19, 20].

As its name suggests, there are two functional programming phases in MapReduce framework: *map* phase and *reduce* phase. The input of the computation is a set of (key, value) pairs and the output of the computation is also a set of (key, value) pairs. We define these (key, value) pairs using angle brackets <*k, v*> on both phase. The key is used primarily in the reduce phase, to determine which values are combined together. The values describe arbitrary information. These are specified using two functions:

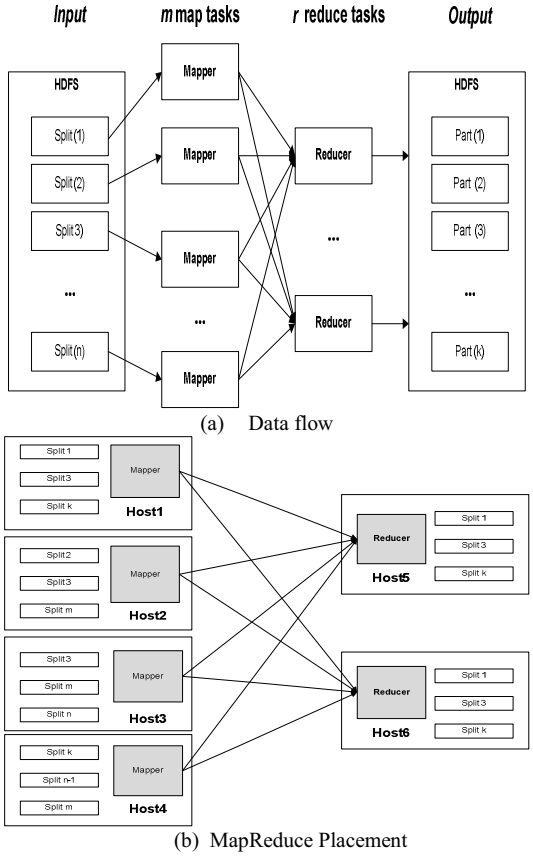
Mapper: <*k1, v1*> → list <*k2, v2*>

Reducer: <*k2*, list<*v2*>> → list<*k3, v3*>

Figure 1 shows the MapReduce computation framework. The computation starts from a map phase in which the map functions are executed in parallel with various splits of the input data which are stored in a distributed file system (DFS). Processing each *split* is assigned to one map task. The output pairs of each map function are hash-partitioned on the intermediate key. Each partition are sorted and then merged in the sorted order by their keys. All the partitions which shared the same key are sent to a single reduce task in which the reducer function obtains the final results. The output of each reduce function is written across the cluster in DFS.

Besides the map and reduce phase, developers are also allowed to program a combine phase which is processed between map phase and reduce phase but executed on the same node as the mapper function. The aim of the combined function is to reduce the communication load through the network among clusters by operating the local pairs.

In the open source community, the Hadoop uses the same architecture of MapRedue as the computation model but implemented in Java, which is used in our research. Programs written in this function style are automatically parallelized and implemented on a cluster of machines. For the MapReduce jobs in Hadoop, a jobtracker acted as a MasterNode splits a job data into several pieces as the input of map phase, and the tasktrackers acted as a DataNode store the intermediate results of map functions in local distributed file system named Hadoop Distributed File System (HDFS). Hadoop schedules the MapReduce computation based on locality and improves the efficiency of overall I/O throughput in the cluster to reduce the computation cost.



(a) Data flow

(b) MapReduce Placement

Figure1. MapReduce Computation Framework

B. Item-based Collaborative Filtering (CF) Algorithm

The purpose of the CF algorithms is to recommend new items or predict the probability of a certain kind of items for a target user based on the user's

previous preferences or the likeness of other users who have the similar preference with the target user. A typical CF operates a set of m users $U=\{u_1,u_2...u_m\}$, and a set of n items $I=\{i_1,i_2...i_n\}$. Each user $u_i \in U$ gives a rating value sets $r_{ij} \in R$ for a corresponding subset of items $I_j \in I$. In our research, the rating value domain R is the set of the integers from 1 to 5. Note that it is possible for r_{ij} to be a null-set. We can divide CF algorithm into two steps as follows:

- Similarity computation. It is to find neighborhood for target u_a or I_a based on the similarity of their rating values. There are several various approaches to computing similarity, such as Pearson correlation, cosine distance.
- Prediction and Recommendation. Once the most similar users or items are found, selecting some new items or users with high predicate value and recommending it.

CF algorithm has two kinds of presentation forms: User-based CF and Item-based CF. In our research, we want to deal with the case of item-based CF.

Unlike the user-based CF algorithm, the item-based CF approach checks into a set of items by which the active user has rated and computes the similarity between them and the target item i , then select the k nearest neighborhoods items. A set of corresponding similarity is denoted as $S=\{s_{i1},s_{i2}...s_{ik}\}$. Once the most similar items are selected, the prediction is then computed. (One fundamental difference between the similarity computation in user-based CF and item-based CF is that in case of user-based CF the similarity is computed along the row of the matrix but in case of the item-based CF the similarity is computed along the columns, (i.e. each pair in the co-rated set corresponds to a different user.)

One critical step in the item-based CF algorithm is to compute the similarity between items and then to select the most similar items. The basic idea in the similarity computation between item i and item j is to segregate the users who have rated both of these items and then to apply a similarity computation approach to determine the similarity s_{ij} . In our research, we use Pearson correlation technique to measure similarity between two items i and j . Let the set of users who both rated items i and j are denoted by U , then the correlation similarity is given by

$$S_{i,j}=\frac{\sum_{u\in U}(R_{u,i}-\bar{R}_i)(R_{u,j}-\bar{R}_j)}{\sqrt{\sum_{u\in U}(R_{u,i}-\bar{R}_i)^2}\sqrt{\sum_{u\in U}(R_{u,j}-\bar{R}_i)^2}} \quad (1)$$

where $R_{u,i}$ denotes the rating of user u on item i , $\bar{R}_i$ is the average rating of the i -th item.

Once the set of most similar items based on the similarity measures are generated, the next step is to compute the prediction on an item i for a user j by computing the sum of the ratings given by the user on the items similar to i . Each ratings is weighted by the corresponding similarity $S_{i,j}$ between items i and j . Here we use a weighted average approach to measure the prediction $P_{u,i}$ as demonstrated in equation 2 [17].

$$P_{u,i}=\bar{R}_i+\frac{\sum_{u\in U}(R_{u,j}-\bar{R}_{\mathrm{u}})S_{i,j}}{\sum_{u\in U}S_{i,j}} \quad (2)$$

III. SCALING-UP ITEM-BASED CF ALGORITHM ON MAPREDUCE

In this section we study how the item-based CF algorithm is implemented on MapReduce and what kind of partition strategies are used to maximize data computation with locality and parallelism.

As equations (1) and (2) shown above, the computation of item-based CF algorithm requires intensive computation power that grows with both the number of users and the number of items. In our method, we partition the three most intensive computations in the item-based CF algorithm into four MapReduce phases. The three parts of intensive computation are: (1) computing the average rating for each item; (2) computing the similarity between item pairs; (3) computing predicted items for the target user. The whole computation flowchart on MapReduce is executed into four phases as Figure 2 shows. Map-I and Reduce-I computes the average rating (in subsection Ⅲ. A); Map-II and Reduce-II computes the similarity (in subsection Ⅲ. B); Map-III and Reduce-III records the similarity matrix preparing for the prediction items computation in Map-IV Reduce-IV (in subsection Ⅲ. C).

A. Computing the average rating for each item

We define $R_{i,j} \in R$ as the rating value on item i by user j . Because the matrix R is sparse, it is represented as a tuple $\Re < i, j, R_{i,j} >$ which

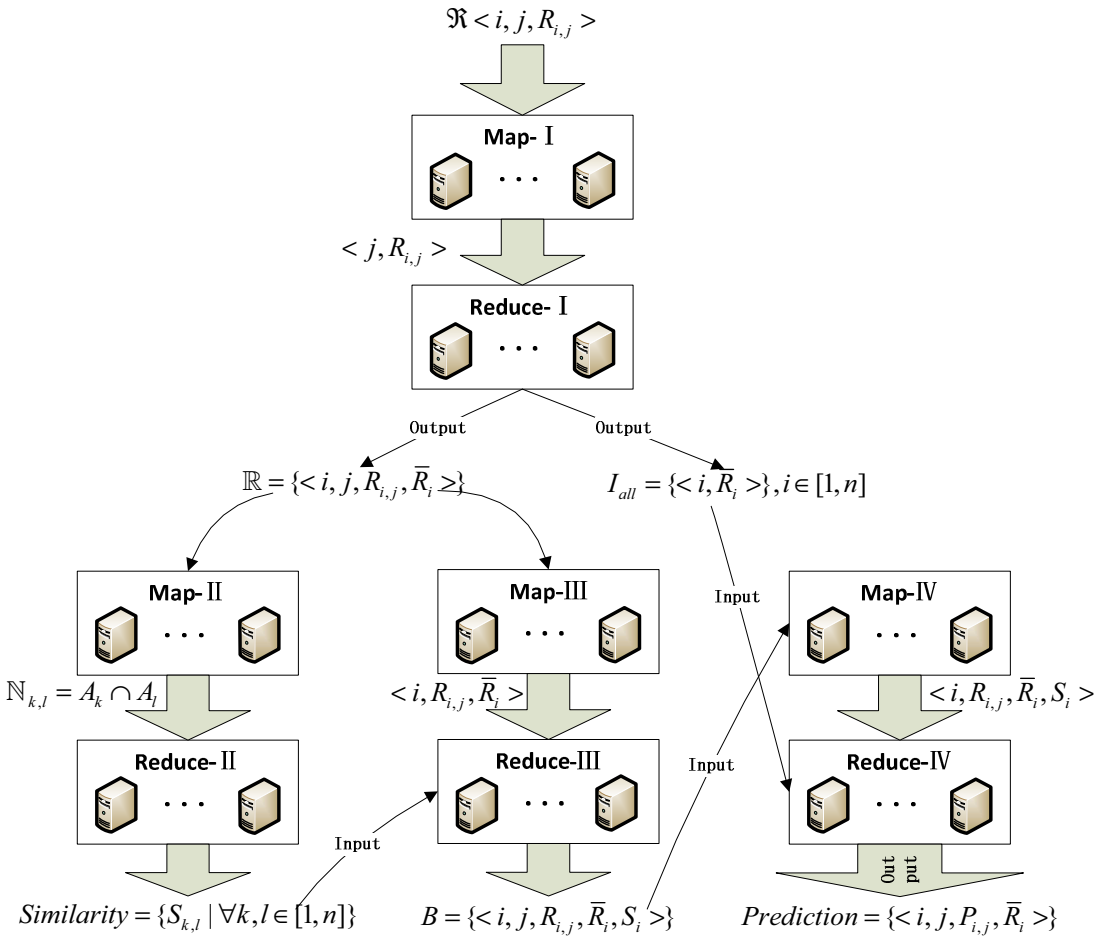


Figure 2 Entire computation flowchart of item-based CF algorithm on MapReduce

records non-null the rating values and the corresponding item i and user j. Then the average rating of each item:

$$\overline{R}_i = \frac{\sum_{j=1}^{m} R_{i,j}}{\sum_{k=1}^{l} 1}; \forall i \in [1, n]$$

where l indicates the number of items the given user rated.

We also denote a tuple $I_{all} < i, \overline{R}_i >, i \in [1, n]$ to record all items and their average ratings.

The computation of average ratings of each item can be implemented by the following MapReduce operations.

Map-I: Map $< i, j, R_{i,j} >$ on i such that the tuple with the same i are shuffled to the same machine in the form of $< j, R_{i,j} >$. If there are n items and m users, and there are k Mappers, the complexity of algorithm in each mapper is $O(\frac{nm}{k})$.

Reduce-I: Take $< j, R_{i,j} >$ and emit $< i, j, R_{i,j}, \overline{R}_i >$ for each $< i, j, R_{i,j} > \in \Re$. At the same time, reducer could record all the items in a single file $I_{all} = \{< i, \overline{R}_i >\} \forall i \in [1, n]$. The two outputs of tuples are all ordered by i. If there are k Reducers, the complexity of the algorithm in each reducer is $O(\frac{nm}{k})$.

The output from Reduce-I will replace the tuple $\Re < i, j, R_{i,j} >$ by tuple $\mathbb{R} < i, j, R_{i,j}, \overline{R}_i >$. In fact, it is not redundant for the storage of average rating in each

tuple $< i, j, R_{i,j}, \overline{R}_i >$, because there are many calculations about $R_{i,j} - \overline{R}_i$ in the next steps.

This phase computes the average rating for each item. We output this information with two formats. We add the average rating in the existing rating record $< i, j, R_{i,j}, \overline{R}_i >$ which is used to calculate the similarity (in subsection Ⅲ.B). We also record an overall item list with the average rating $< i, \overline{R}_i >$ which is used to calculate the prediction rating (in subsection Ⅲ.C).

B. Computing Similarity

In this step, we could calculate the similarity between two arbitrarily items and form the similarity matrix. We denote the $S_{i,j}$ as the similarity between item i and item j and the similarity matrix as $S = \{S_{i,j} \mid \forall i, j \in [1,n]\}$.

We also define $A_i = \{< i, j, R_{i,j}, \overline{R}_i > \mid \forall \mathrm{j} \in [1,n], < i, j, R_{i,j}, \overline{R}_i > \in \mathbb{R}\}$ as a subset of users who rated the item i, redefine the matrix $\mathbb{N}_{\mathrm{k},l} = A_k \cap A_l, \forall k, l \in [1,n]$, where $\mathbb{N}_{k,l}$ means the set of users who rated the item k and item l. In order to obtain good scalability, all the data required in the calculation of $\mathbb{N}_{k,l}$ should be stored in the same node. Each computation of $\mathbb{N}_{k,l}$ could be completed in the same node, while each calculation of $A_{i,j}$ should be distributed to different nodes in MapReduce cluster. From the equation (1), we know the computation of $S_{i,j}$ could get all the required data from $\mathbb{N}_{i,j}$, and it could run on the local.

Map-II: Map $\{A_i \mid i \in [1,m]\}$ to $\{\mathbb{N}_{k,l} \mid \forall k, l \in [1,n]\}$. If there are k Mappers, the complexity of algorithm in each mapper is $O(\frac{n^2 m}{k})$.

Reduce-II: Take $\{\mathbb{N}_{k,l} \mid \forall k, l \in [1,n]\}$, and emit $\{S_{k,l} \mid \forall k, l \in [1,n]\}$. If there are k Reducers, the complexity of algorithm in each Reducer is $O(\frac{n^2 m}{k})$.

This phase computes the similarity between item pairs for the users. This tuple will be used to calculate the prediction rating (in subsection Ⅲ. C).

We could find that the n (number of items) has more influence in the computation workload. The item-based algorithm is more suitable to the situation $m >> n$ that means it could solve explosion problem of the number of users. As the tuples $\mathbb{R} < i, j, R_{i,j}, \overline{R}_i >$ was ordered by the items id in the Reduce-I, the tuples, which belong to the same item, could be arranged into the same node easily.

C. Computing Prediction Matrix

We transfer the matrix $\mathbb{R}$ to U which sorted by user id:

$$U = \begin{vmatrix} \begin{bmatrix} < 1,1,R_{1,1},\overline{R}_1 > \\ \ldots\ldots \\ < i,1,R_{i,1},\overline{R}_i > \\ \ldots\ldots \end{bmatrix} \\ \ldots\ldots \\ \begin{bmatrix} < 1,j,R_{1,j},\overline{R}_1 > \\ \ldots\ldots \\ < i,j,R_{i,j},\overline{R}_i > \\ \ldots\ldots \end{bmatrix} \\ \ldots\ldots \end{vmatrix} = \begin{vmatrix} U_1 \\ U_2 \\ \ldots \\ U_j \\ \ldots \\ U_m \end{vmatrix} = \{U_j \mid \forall j \in [1,m]\}$$ whe

where

$$U_j = \left\{< i, j, R_{i,j}, \overline{R}_i > \mid \forall i \in [1,n], < i, j, R_{i,j}, \overline{R}_i > \in \mathbb{R}\right\}$$

and U_j means the set of rating tuples by user j.

We define the similarity matrix as

$$S = \begin{vmatrix} S_{11}\ldots\ldots S_{1n} \\ \ldots\ldots \\ S_{i1}\ldots\ldots S_{in} \\ \ldots\ldots \\ S_{n1}\ldots\ldots S_{nn} \end{vmatrix} = \begin{vmatrix} S_1 \\ \ldots \\ S_i \\ \ldots \\ S_n \end{vmatrix}$$ and

$S_i = \{S_{i,j} \mid \forall j \in [1,n]\}$;

S_i means the set of similarity between item i and another arbitrary item j.

Map-III: Map $\mathbb{R} < i, j, R_{i,j}, \overline{R}_i >$ on j such that tuples with the same j are shuffled to the same machine in the form of $< i, R_{i,j}, \overline{R}_i >$.

If there are k Mappers, the complexity of algorithm in each Mapper is $o(\frac{nm}{k})$.

Reduce-III: Take $< i, R_{i,j}, \overline{R}_i >$ and S, then emit $B =< i, j, R_{i,j}, \overline{R}_i, S_i >$ for each $< i, j, R_{i,j} > \in \Re$. The tuples are ordered by the user id j.

If there are k Reducers, in each machine, the complex of algorithm is $o(\frac{nm}{k})$.

This phase prepares the data for the next step of prediction calculation. It integrates the existing rating record, average rating for each item and similarity tuples for each item.

Define the tuple $B =< i, j, R_{i,j}, \overline{R}_i, S_i >$ by which, we compute the prediction $P_{i,j}$ to item i by user j according to equation (2). The matrix of prediction P is defined as

$$P = \begin{vmatrix} \{P_{1,1}, P_{3,1}, P_{5,1}\} \\ ..... \\ \{P_{2,j}, P_{3,j}, P_{4,j}, P_{7,j}\} \\ ..... \\ \{P_{1,m}, P_{2,m}\} \end{vmatrix} = \begin{vmatrix} P_1 \\ ..... \\ P_j \\ ..... \\ P_m \end{vmatrix}$$

where

$$P_j = \{P_{i,j} \mid \forall i \in [1,n], < i, j, R_{i,j}, \overline{R}_i > \notin \mathbb{R}\}; \forall j \in [1,m]$$

P_j means the set of prediction value of item i by on arbitrary user j who never rated the item i.

Map-IV: Map $B =< i, j, R_{i,j}, \overline{R}_i, S_i >$ on j such that tuples with the same j are shuffled to the same machine in the form of $< i, R_{i,j}, \overline{R}_i, S_i >$.

If there are k Mappers, the complexity of algorithm in each Mapper is $o(\frac{nm}{k})$.

Reduce-IV: Take $< i, R_{i,j}, \overline{R}_i, S_i >$ and I_{all} (which is the output of the Reducer-III), then emit $< i, j, P_{i,j}, \overline{R}_i >$ for each $< i, j, R_{i,j} > \in \Re$. The tuple is ordered by the key j and $P_{i,j}$.

If there are k Reducers, in each machine, the complex of algorithm is $o(\frac{nm}{k})$.

In the last phase, the prediction rating for all the users to all the items is calculated. Then the recommendation list can be obtained.

In summary, throughout the four Map-Reduce phases, the item-based CF algorithm has been implemented in parallel on MapReduce, by splitting the data file with same userID or itemID into the same Maper or Reducer. The partition strategies ensure the local computation. In each Map-Reduce phase, the complexity of the algorithm for each node can be controlled by the number of Mapper and Reducer (k) when the data size increases.

IV. EXPERIMENTATION AND EVALUATION

In this section we investigate the performance of the item-based CF algorithm on our Hadoop cluster with a particular focus on its scalability and efficiency. Scalability and efficiency are the most used metrics in performance evaluation, because an algorithm that is not scalable well will be of limited utility in practice.

A. Setup

We performed all experiments on a Hadoop cluster which includes 3 nodes, one node as MasterNode and other 2 nodes as DataNodes. These nodes are common PC machines with Intel P4 CPU, 1G RAM, 80G disk and all running Ubuntu Linux 10. All the machines were connected with one 100Mbps switch.

In our research, each node could support maximum of five Map or Reduce tasks and we set the same numbers of map and reduce task. Because the map and reduce task cannot execute at the same time, our platform could support 10 map tasks and 10 reduce tasks. We record the number of map or reduce tasks as the number of nodes. For example, we use 20 map tasks and 20 reduce tasks in one experiment, then we record the number of nodes is 20.

In the experiments we use the MoveLens [19-21] datasets which consists of movie rating data collected using a web-based research recommendation system. The datasets contains 943 users, 1670 movies item and about 54,000 ratings on a scale from 1 to 5. As the algorithm accuracy and recall are not considered in this experiment, we could produce the data based on the Movielens' data format. When we increased the data size, we also increased the number of users to keep the same ratio of sparse data. Therefore, the computation load of algorithm could keep the liner increased.

B. Performance Evaluation

To measure the performance in terms of the scalability and efficiency of parallel algorithms, there are several performance metrics such as speedup, scaled speedup, sizeup, experimentally determined serial fraction and isoefficiency function [22]. Isoefficiency and speedup are two useful scalability metrics [23]. The former evaluates

the performance of an algorithm-machine combination through modeling an isoefficiency function. The later evaluates the execution performance change of a fixed size problem as the number of processors increases. In our experiment, we use the speedup and isoefficency as the performance metric.

(1) Speedup

Amdahl's law [24] roughly models the performance of speedup S_p,

$$S_p = \frac{T_1}{T_p} \tag{3}$$

where T_1 is the amount of sequential execution time with single processer. T_p is the amount of parallel execution time with p processor. If the algorithm is scalable, the speedup has a linear relation with the numbers of nodes with the datasize fixed. From Figure 3 we can see that the speedup increased relative linearly with the growth of the number of nodes with the different fixed datasize (10K, 100K, 1M, 10M). We also find that the larger fixed datasize obtained a better speedup. Thus the Hadoop demonstrates that Hadoop has better scalability when it processes large datasets.

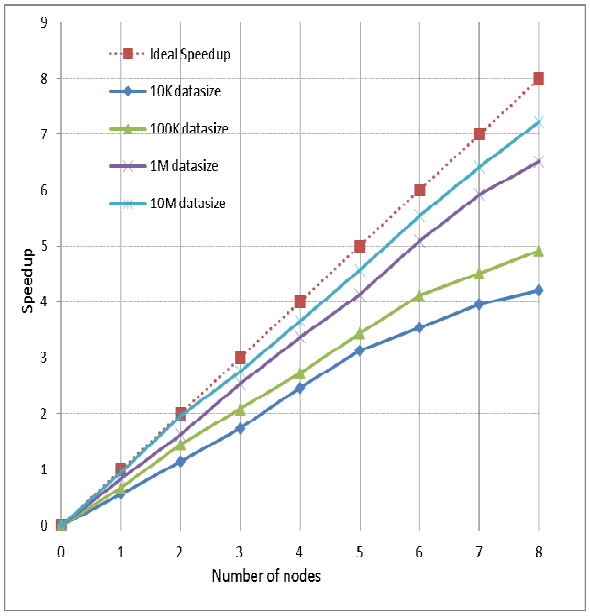


Figure 3 Speedup of item-based CF algorithm

(2) Isoefficiency

Kumar *et al.* [25] proposed the isoefficiency concept which measures how much work must be increased to keep the efficiency unchanged. The isoefficiency could be expressed

$$E_p = \frac{S_p}{p} = \frac{T_1}{T_p \times p} = \frac{T_1}{T_1 + T_0} = \frac{1}{1 + T_0 / T_1} \tag{4}$$

where T_0 is the time of communication cost in parallel process. If the algorithm is scalable, it should keep E_p constant. For the fixed E_p, $T_1 = f_E(P) = kP$ and $f_E(P)$ should be a linear function. That is $T_1 = T(m) = lm$ which means the size of data m has linear relation with the number of nodes P. Therefore, the number of nodes P has a liner relation with the size of data m. i.e $P = \frac{l}{k} m$.

Thus if P (the number of nodes) has a linear relation with m (the datasize), it means that the algorithm is scalability. As showed in Figure 4, setting the fixed execution time (10 mins, 20 mins, 30 mins), the number of nodes has a good linear increase with the datasize growing. We also find that the longer fixed execution time has a better performance.

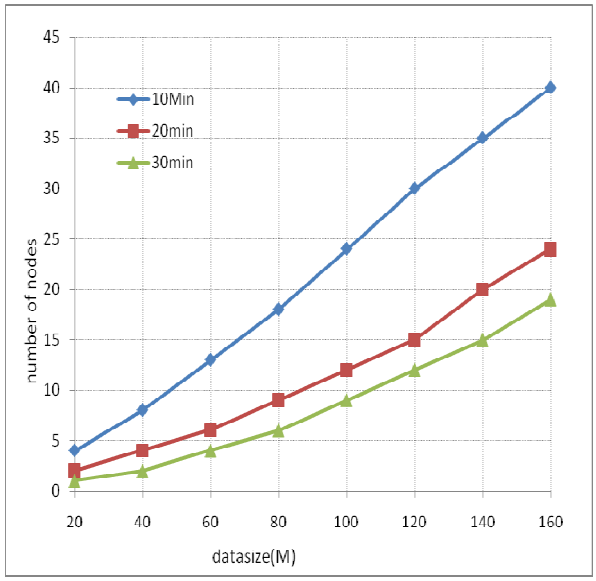


Figure 4 Efficiency function of number of nodes and datasize in fixed running-time

In summary, the experimental results show a good scalability and efficiency of this algorithm on our Hadoop cluster.

V. CONCLUSIONS

In this paper, we parallelized a scaling-up item-based CF algorithm on MapReduce, by splitting the whole data

file with same userID or same itemID into the same Maper or Reducer. To minimize the communication cost, we used effective partition strategies to realize the local computation in each Map-Reduce phase. The experiment results show that it is obtained good scalability and efficiency performance of our proposed scale-up item-based CF algorithm executed in Hadoop cluster. Due to the limited experimental resource, we just use 3 nodes in our Hadoop cluster. In the future work, we will do our experiment on a larger Hadoop cluster on which the experiment results may show more useful information.

There are still some future works we will do. On the algorithmic side, the first priority is to optimize the resource allocation and execution process in MapReduce. On the application side, we will explore other applications, which are enabled by item-based CF algorithm with good scalability, to process larger scale data.

ACKNOWLEDGMENT

The work presented in this paper was supported by the Australian Research Council (ARC) under discovery grant DP110103733.

REFERENCES

[1] J. Dean and S. Ghemawat, "Mapreduce:Simplified data processing on large clusters," *Procedings of OSDI 2004: Sixth Sixth Symposium on Operating System Design and Implementation,* 2004.

[2] S. Ibrahim, H. Jin, L. Lu, L. Qi, S. Wu, and X. H. Shi, "Evaluating MapReduce on Virtual Machines: The Hadoop Case," in *Cloud Computing, Proceedings*, 2009, pp. 519-528.

[3] J. Ekanayake, S. Pallickara, and G. Fox, "MapReduce for Data Intensive Scientific Analyses," in *eScience '08. IEEE Fourth International Conference on*, 2008, pp. 277-284.

[4] M. Bhandarkar, "MapReduce programming with apache Hadoop," in *Parallel & Distributed Processing (IPDPS), 2010 IEEE International Symposium on*, 2010, pp. 1-1.

[5] *Hadoop*. Available: http://hadoop.apache.org/core/.

[6] J. Leverich and C. Kozyrakis, "On the energy (in)efficiency of Hadoop clusters," *SIGOPS Oper. Syst. Rev.,* vol. 44, pp. 61-65, 2010.

[7] T. White, *Hadoop: The Definitive Guide*: O'Reilly Media, Inc., 2009.

[8] Z. D. Zhao and M. S. Shang, "User-Based Collaborative-Filtering Recommendation Algorithms on Hadoop," in *International Workshop on Knowledge Discovery and Data Mining*, 2010, pp. 478-481.

[9] T. HengSong and Y. HongWu, "A Collaborative Filtering Recommendation Algorithm Based on Item Classification," in *Circuits, Communications and Systems. PACCS '09. Pacific-Asia Conference on*, 2009, pp. 694-697.

[10] G. Song-Jie, Y. HongWu, and T. HengSong, "Combining Memory-Based and Model-Based Collaborative Filtering in Recommender System," in *Circuits, Communications and Systems. PACCS '09. Pacific-Asia Conference on*, 2009, pp. 690-693.

[11] S. Xiaoyuan, T. M. Khoshgoftaar, and R. Greiner, "Imputed Neighborhood Based Collaborative Filtering," in *Web Intelligence and Intelligent Agent Technology, 2008. WI-IAT '08. IEEE/WIC/ACM International Conference on*, 2008, pp. 633-639.

[12] S. Ping and Y. HongWu, "An Item Based Collaborative Filtering Recommendation Algorithm Using Rough Set Prediction," in *Artificial Intelligence, 2009. JCAI '09. International Joint Conference on*, 2009, pp. 308-311.

[13] G. SongJie, "Joining Case-Based Reasoning and Item-Based Collaborative Filtering in Recommender Systems," in *Electronic Commerce and Security, 2009. ISECS '09. Second International Symposium on*, 2009, pp. 40-42.

[14] W. Pu and Y. HongWu, "A Personalized Recommendation Algorithm Combining Slope One Scheme and User Based Collaborative Filtering," in *Industrial and Information Systems, 2009. IIS '09. International Conference on*, 2009, pp. 152-154.

[15] R. M. Bell and Y. Koren, "Scalable Collaborative Filtering with Jointly Derived Neighborhood Interpolation Weights," in *Data Mining, 2007. ICDM 2007. Seventh IEEE International Conference on*, 2007, pp. 43-52.

[16] C. F. Lai, J. H. Chang, C. C. Hu, Y. M. Huang, and H. C. Chao, "CPRS: A Cloud-Based Program Recommendation System for Digital TV Platforms," *Advances in Grid and Pervasive Computing, Proceedings,* vol. 6104, pp. 331-340, 2010.

[17] Badrul Sarwar, George Karypis, Joseph Konstan, and John Reidl, "Item-based collaborative filtering recommendation algorithms," in *Proceedings of the 10th international conference on World Wide Web*, Hong Kong, Hong Kong, 2001, pp. 285-295.

[18] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," *Commun. ACM,* vol. 51, pp. 107-113, 2008.

[19] C.-T. e. a. Chu, "Mapreduce for Machine Learing on Multicore," *NIPS 2006,* 2006.

[20] W. Z. Zhao, H. F. Ma, and Q. He, "Parallel K-Means Clustering Based on MapReduce," *Cloud Computing, Proceedings,* vol. 5931, pp. 674-679, 2009.

[21] B. N. Miller, I. Albert, S. K. Lam, J. A. Konstan, and J. Riedl, "MovieLens unplugged: experiences with an occasionally connected recommender system," presented at the Proceedings of the 8th international conference on Intelligent user interfaces, Miami, Florida, USA, 2003.

[22] U. Ramachandran, Venkateswaran H., Sivasubramaniam Anand, and S. Aman, "Issues in understanding the scalability of parallel systems," in *In Proceedings of the First Inernational Workshop on Parallel Processing*, Bangalore, India, 1994, pp. 399-404.

[23] X. Zhang, Yan Yong, and Ma Qian, "Measuring and analyzing parallel computing scalability," in *International conference on parallel processing*, 1994, pp. 295-303.

[24] G. Amdahl, "Validity of the sigle-processor approach to achieving large scale computing capabilities," in *Proc. AFIPS Conf.*, 1967, pp. 483-485.

[25] V. Kumar and V. Singh, "Scalability of parallel algorithms for the all-pairs shortest path problem: A summary of results," in *Proc. of Conf. on Parallel Processing*, Chicago, IL, 1990, pp. 136-140.