MapReduce-Based SimRank Computation and Its Application in Social Recommender System

Lina Li, Cuiping Li, Hong Chen, Xiaoyong Du
Key Laboratory of Data Engineering and Knowledge Engineering, MOE
Renmin University of China,Beijing, China
{*lina, licuiping, chong, duyong*}*@ruc.edu.cn*

***Abstract*—Recently there has been a lot of interest in graph-based analysis, with examples including social network analysis, recommendation systems, document classification and clustering, and so on. A graph is an abstraction that naturally captures data objects as well as relationships among those objects. Objects are represented as nodes and relationships are represented as edges in the graph. There are many cases in which similarities among nodes are required to compute. SimRank is one of the simple and intuitive algorithms for this purpose. It is rigidly based on the random walk theorem. Existing methods on SimRank computation suffer from one limitation: the computing cost can be very high in practice. In order to optimize the computation of SimRank, a few techniques have been proposed.**

However, the performance of these methods are still limited by the processing ability of the single computer. Ideally, we would like to develop new parallel solutions that can offer improved processing power to compute SimRank on large data set. In this paper, we propose parallel algorithms for SimRank computation on Map-Reduce framework, and more specifically its open source implementation, Hadoop. Two different parallel methods are proposed and their performances are evaluated and compared. Furthermore, we employ the proposed methods to do the similarity computation in order to recommend appropriate products to users in social recommender systems.

***Keywords*-Simrank; Parallel; Mapreduce; Recommendation**

I. Introduction

Recently there has been a lot of interest in graph-based analysis, with examples including social network analysis, recommendation systems, document classification and clustering, and so on. A graph is an abstraction that naturally captures data objects as well as relationships among those objects. Objects are represented as nodes and relationships are represented as edges in the graph.

There are many cases in which it would be important to answer queries such as "Which two nodes are the most similar in the network?". To this end, a great number of similarity measures[14], [16], [18], [12], and [19] have been proposed.

Typically there are two kinds of methods computing the similarity between two nodes. One method is based on the content of text. This method takes the text object as a set of short terms or phrases, and the weights of terms constitute a feature vector. Then proper methods are used to compute the similarity between the text objects with the feature vector. The other method is used to compute similarity between linked objects. As the objects are linked, more information can be inferred from the relation between the two objects. SimRank is one of the methods to compute similarity between linked objects. And SimRank is applied in many fields such as collaborative filtering recommendation, cluster algorithm, classification algorithm, and search engines.

Existing methods on SimRank computation suffer from one limitation: the computing cost can be very high in practice. In order to optimize the computation of SimRank, a few techniques have been proposed [13], [22]. However, the performance of these methods are still limited by the processing ability of the single computer. Ideally, we would like to develop new parallel solutions that can offer improved processing power to compute SimRank on large data set.

In this paper, we propose parallel algorithms for SimRank computation on Map-Reduce framework [34], and more specifically its open source implementation, Hadoop [35]. Two different parallel methods are proposed and their performances are evaluated and compared. Furthermore, we employ the proposed methods to do the similarity computation in order to recommend appropriate products to users in social recommender systems.

Specifically, this paper has made the following contributions.

- We propose two Map-Reduce-based computation algorithms for the initial iterative SimRank computation and the matrix multiplication SimRank computation respectively.
- We employ the proposed methods to do the similarity computation in order to recommend appropriate products to users in social recommender systems.
- We conduct extensive experiments on various kinds of real data sets to verify the efficiency and effectiveness of the proposed methods.

The rest of this paper is organized as follows. Section II gives the background information of our study. Section III introduces our parallel SimRank computation techniques. A performance analysis of our methods is presented in Section IV. Section V gives the recommendation policies in social

978-0-7695-5006-0/13 $26.00 © 2013 IEEE
DOI 10.1109/BigData.Congress.2013.26

recommender systems. We discuss related work in Section VI and conclude the study in Section VII.

II. PRELIMINARIES

SimRank is an important method to compute similarities between objects. This method can be applied to compute the similarity between hyper-link documents, and then sort the documents by the similarity value. This method can also be used in recommendation system to recommend an item to a client who might be the potential costumer.

Typically, there are two ways to compute SimRank. The first one is a naive formula which is raised by the SimRank proposer. This method calculates similarity accurately, but the drawback of this method is the high time complexity of $O(n^3)$ for each iteration step and the high space complexity of $O(n^2)$. The second one is the matrix multiplication method. This method turns the initial formula into a standard adjacent matrix multiplication.

In this section, we provide the necessary background for the subsequent discussions. We first present some notations and assumptions that are adopted in this paper in Section II-A, and then give a brief review of two SimRank computations in Section II-B and II-C.

A. Notations and Assumptions

Table I lists the main symbols we use throughout the paper.

Table I
SYMBOLS

Symbol	Definition and Description
A	matrices (bold upper case)
A^T	transpose of matrix A
$A(i,j)$	(i,j)-th element of matrix A
n	the number of nodes in the graph
k	the current iterative step
c	the decay factor for SimRank

Without loss of generality, given a graph $G = (V, E)$ where nodes in V represent objects of the domain and edges in E represent relationships between objects. For a node v in a graph, $I(v)$ and $O(v)$ denote the set of in-neighbors and out-neighbors of v, respectively.

B. Initial Iterative Algorithm

This method is based on the following assumption: if two objects are similar, then the neighbors of the objects are also similar. Let $S(a,b) \in [0,1]$ denote the similarity between two objects a and b, the iterative similarity computation equation of SimRank is as follows:

$$S(a,b) = \frac{c}{|I(a)||I(b)|} \sum_{i=1}^{|I(a)|} \sum_{j=1}^{|I(b)|} S(I_i(a), I_j(b)) \qquad (1)$$

where c is the decay factor for SimRank (a constant between 0 and 1), $I(a)$ is a set of the nodes which are input nodes of a, $I_j(a)$ represents node j in the node set $I(a)$. This formula indicates that the similarity of nodes are mainly determined by their neighbors. Initially, the similarity between a node and itself is 1, and the similarity between two different nodes is 0. Later, similarities are computed iteratively using the following iterative formula:

$$S^k(a,b) = \frac{c}{|I(a)||I(b)|} \sum_{i=1}^{|I(a)|} \sum_{j=1}^{|I(b)|} S^{k-1}(I_i(a), I_j(b)) \qquad (2)$$

C. Matrix Multiplication Algorithm

Paper [21] gives a new way to compute SimRank scores between objects. In this method, the initial formula is turned into the following matrix format:

$$S^k = c * W^T S^{k-1} W + (1-c) * I \qquad (3)$$

where I is an identity matrix, W is a column standardized matrix of the adjacency matrix of the graph. W^T is the transport matrix of W. Similarity computation of objects in this method is changed to do some matrix multiplications. And the related proof is presented in the original paper.

D. Map-Reduce Computing Model

Map-Reduce is a parallel computing model. It splits the input data set into M parts and starts M mappers, and each mapper deals with one data part. Mapper produces key/value pairs and writes them into the intermediate files. If there are N key values in the intermediate result, then N reducers need to be started to obtain the final result. This is called a Map-Reduce job. Designing a Map-Reduce algorithm has some challenges, such as any computing process should be able to turn into map/reduce procedures and obey strict rules. The complexity of the algorithm makes it more challenging. So far, there are some modified frameworks such as Pig Latin [36], SCOPE [6]. They provide a higher level language and even other operators such as join operator. Frameworks like Hadoop [35] and iMapReduce [23] are all modified on the initial Hadoop. In this paper, we don't make any change on the Hadoop platform, but we take the strategy that iMapReduce uses which will be mentioned in the following sections. All implementations are based on the initial Hadoop platform.

E. Work of This Paper

Existing methods are not scalable and not practical on large dataset. This paper pays attention to the details of the parallel SimRank computation on Hadoop platform. The following sections present detailed descriptions about the algorithms, applications, and experimental evaluations on different scale data sets. Accuracies and efficiencies of proposed methods are tested and presented.

III. SimRank Parallel Computation

Equation 2 and 3 show that SimRank scores are propagated through the graph in multiple iterations until convergence. As discussed earlier, this computation framework is very expensive in most real applications. In this section, we introduce two new SimRank computation methods which utilize the parallel computation power of Map-Reduce to speed up SimRank computations on large graphs.

A. Parallelism of Initial Iterative Algorithm

Initial iterative algorithm performs same operations on a data set for several iterations. Here we describe our proposed algorithm for the SimRank computation under Map-Reduce framework. we first re-write Equation 2 into the following form:

$$S^k(a,b) = \sum_{i=1}^{|I(a)|} \sum_{j=1}^{|I(b)|} \frac{c}{|I(a)||I(b)|} S^{k-1}(I_i(a), I_j(b)) \quad (4)$$

Let $i \in I(a)$, $j \in I(b)$, we have:

$$S^k(a,b) = \sum_{i=1}^{|I(a)|} \sum_{j=1}^{|I(j))|} \frac{c}{|I(a)||I(b)|} S^{k-1}(i,j) \quad (5)$$

Let $S_{ij}^{k-1}(a,b) = \frac{c}{|I(a)||I(b)|} S^{k-1}(i,j)$, then we have:

$$S^k(a,b) = \sum_{i=1}^{|I(a)|} \sum_{j=1}^{|I(b))|} S_{ij}^{k-1}(a,b)(i \epsilon I(a), j \epsilon I(b)) \quad (6)$$

where $S_{ij}^{k-1}(a,b)$ represents the similarity contribution of node pair (i, j) making to node pair (a, b) in the k^{th} iteration. $S(a,b)$, the similarity of node pair (a,b), is the sum of all contributions made by other node pairs during one iteration.

During one iteration, all contributions made by the node pair (i,j) to other node pair (a,b) are computed as $S_{ij}(a,b)$. This process can be achieved by a parallel map procedure. The key of the map output is $(a,b)(a < b)$, and the output record is $(< a, b >, S_{ij}(a,b))$.

With the features of Map-Reduce Computing Model, the records with the same key will be pushed to the same reducer. The reduce procedure then sums up all the contributions for the pair. In the k^{th} iteration step, since the similarities $S^{k-1}(i,j)$ of any node pair for the last iteration is known, the similarity of $S_{ij}^k(a,b)$ is easy to be computed for the k^{th} iteration step. One iteration step uses one Map-Reduce job.

This is a typical iterative algorithm. The computation tasks of each iteration are independent and similar. They need read data from or write data into the file system. So the major challenge for an efficient algorithm for this is to minimize the I/O and communication cost.

In our setting, the topology of the whole graph , especially the input and output node collections of a node need to be obtained during the computation. In order to improve the computation efficiency, we consider to put this information in the cache.

The input and output information is recorded as $(uin : a, b, \ldots | out : x, y, \ldots)$. It is written in a file and then sent to each computer. Since the graph is static, the information of the input and output node collections is static too. In the implementation, the information is cached in the Distributed Cache to reduce the communicate cost between task tracker and the job tracker. When a Map-Reduce job is run for the first time, the input and output node collections information is put into the memory. It is kept in the cache during the whole computing process and can be used for all reduce procedure. Through this way, the algorithm can reduce the expenses of swap, and reduce the communication cost as well.

Algorithm 1 outlines the pseudo-code of the parallel initial iterative SimRank computation algorithm based on Map-Reduce for the node pair (u, v).

Algorithm 1 Parallel Initial Iterative SimRank Computation

INPUT: The graph $G = \{V, E\}$

OUTPUT: The similarity of the node pair (u, v)

I. Pre-processing

01: Map edge (u, v) into records $(\langle u \rangle, out : v)$ and $(\langle v \rangle, in : u)$

02: Reduce the records with the same key to a record $(\langle u \rangle, in : a, b, \ldots | out : x, y, \ldots)$

II. Iterative Computation

01: Given (u, v, sim), $i \in O(a)$, and $j \in O(b)$

02: Computing $S_{uv}(i,j)$ and recorded as $(\langle i, j \rangle, sim)$

The whole algorithm consists of two phases. The first phase is to do the pre-precessing. During this phase, the topology of the graph is pre-processed. Each graph edge (u, v) is changed into the format of $(\langle u \rangle, in : a, b, \ldots | out : x, y, \ldots)$. In this way, the inputs and outputs of all nodes are obtained. The pre-processing phase is finished in a Map-Reduce computing model. In the Map procedure, the input record (u, v) is turned into the records $(\langle u \rangle, out : v)$ and $(\langle v \rangle, in : u)$. In the Reduce procedure, the records with the same key are reduced to a record $(\langle u \rangle, in : a, b, \ldots | out : x, y, \ldots)$.

In the second phase, iterative SimRank computation is performed. The result files generated in the pre-processing phase are distributed to all computers in the platform. These files will be used in the following iterative computing. The input file of k^{th} iteration step is the output of the $(k-1)^{th}$ iteration.

In the beginning of the second phase, the similarity between a node and itself is 1. So initially the similarity record is like $(u, u, 1)$. In order to do the computation on Hadoop platform, the input file is split into several small files. Later, each map produce will be performed on one

small file. Each iteration will start a Map-Reduce job. The details of the iteration computation is as below:

- Map procedure: in the pre-process phase, the topology of the graph has been stored in every computer. Assuming the input similarity for the $(k-1)^{th}$ iterative step is recorded as (u, v, sim). With records $(\langle u\rangle, in : a, b, \ldots | out : x, y, \ldots)$ and $(\langle v\rangle, in : a, b, \ldots | out : x, y, \ldots)$ cached in the memory, the score of $S^k_{uv}(i, j)$ can be computed. $S^k_{uv}(i, j)$ represents the contribution made by the node pair (u, v) to other node pairs (i, j). The result is recorded in format $(\langle i, j\rangle, sim)$.
- Reduce procedure: the similarity score of node pair (u, v) for this iteration can be computed by summing up those records $(\langle i, j\rangle, sim)$ which share the same key.

B. Parallelism of Matrix Multiplication Method

Equation 3 entails three matrix-matrix multiplications. So the core of this method is to do the matrix multiplication in parallel. We propose two methods, one is the single tuple method, and the other is the row divided method. We will introduce them respectively in the subsequent sections.

1) Single Tuple Method (STM): Let us analyze the matrix multiplication first. Given two matrices A and B, the multiplication of A and B is C: $C(i, j) = \sum_{t=1}^{m} A(i, t)B(t, j)$. Suppose both Matrices A and B have m columns and m rows, and the value of element in i^{th} row and j^{th} column is t. Then the element can be represented as a record $(i, j, t)(t \neq 0)$.

In the pre-processing phase, each element in Matrix A is represented as a record $(j, 1, i, j, a_{ij})$, and each element in Matrix B is represented as a record$(i, 2, i, j, b_{ij})$. The description of the pre-processing phase is as below:

First, turn the record (i, j, a_{ij}) of Matrix A to $(\langle j\rangle, 1, i, j, a_{ij})$; turn the record (i, j, b_{ij}) of Matrix B to $(\langle i\rangle, 2, i, j, b_{ij})$. Then, devide records into two categories: $(\langle j\rangle, 1, i, j, a_{ij})$ and $(\langle i\rangle, 2, i, j, b_{ij})$. The records with same key are sent to the same reducer. In a reduce procedure, the element record $(\langle t\rangle, 1, m, t, a_{mt})$ from matrix A is used as $A(m, t)$, and the element record $(\langle t\rangle, 2, t, n, b_{tn})$ from matrix B is used as $B(t, n)$. The multiplication is performed to obtain $C(m, n) = A(m, t) * B(t, n)$, in which c_{mn} is the element value of the result matrix C, recorded as $(\langle m, n\rangle, c_{mn})$. Then, another Map-Reduce job is started to sum up all c_{mn} with the same key. Next, pre-processing the result Matrix C, which will be used as the input of the next iteration.

Matrix Multiplication method needs two Map-Reduce jobs. One job multiplies each element from Matrix A by an element from Matrix B, and the other job is used to sum them up.

Now it is ready to apply the method to compute the SimRank. The formula of SimRank is: $S^k = c * W^T S^{k-1} W + (1-c) * I$, in which W is the column standardized adjacency matrix, and c is the delay factor.

When computing the SimRank, if let $(c * W^T) be W^T_c$, $(1 - c) * I$ be I_{1-c}, then the formula is turned into $S^k = W^T_c S^{k-1} W + I_{1-c}$. Since Matrix I is represented as a collection of records like $(1, 1, 1)$, Matrix I_{1-c} is represented as a collection of records like $(1, 1, 1 - c)$.

At first, Matrix W^T_c is multiplied by Matrix S^{k-1}, and get the result Matrix T. Then Matrix T is multiplied by Matrix W. The result matrix is added to Matrix I_{1-c}. At this time, one iteration step is finished. Since the graph is static, matrix W^T_c and matrix W in the formula $S^k = W^T_c S^{k-1} W + I_{1-c}$ is unchanged during the computation.

2) Row Divided Method (RDM): In order to further improve the computation efficiency, we develop another method for the matrix multiplication. This method is more effective than the one before.

Let us analyze the matrix multiplication in another way. For example: matrix A can be multiplied by matrix B as below:

$$\begin{pmatrix} A_{11} & A_{12}... & A_{1n} \\ A_{21} & A_{22}... & A_{2n} \\ ... & & \\ A_{m1} & A_{m2}... & A_{mn} \end{pmatrix} * \begin{pmatrix} B_{11} & B_{12}... & B_{1t} \\ B_{21} & B_{22}... & B_{2t} \\ ... & & \\ B_{n1} & B_{n2}... & B_{nt} \end{pmatrix}$$

in which the element $C(i, j)$ of the result Matrix C can be looked as the vector multiplication result of the i^{th} row of Matrix A and the j^{th} column of Matrix B. since each row of Matrix A can be multiplied by different column of B independently, the multiplication can be finished in parallel.

Here we do not divide Matrix B. Each computer in distribute platform keeps all records of Matrix B in memory. The matrix multiplication is performed as following:

- Map procedure: turn the input record (i, j, a_{ij}) into $(\langle i\rangle, j, a_{ij})$.
- Reduce procedure: from input records $(\langle i\rangle, j, a_{ij})$, get the row record in Matrix A as record $(\langle i\rangle, 1, a_{i1}; 2, a_{i2}; 3, a_{i3}; ...)$. Multiplying the row record to each column of the Matrix B.

Experiments show this method is more effective than the single tuple method. This is because the division of Row Divided Method is based on row unit of a matrix, while the division of Single Tuple Method is based on tuple unit of a matrix. Row Divided Method produces less intermediate records than the Single Tuple Method does. Thus it reduces the expenses of sort in shuffle procedure.

In the Single Tuple Method, there are $(m*n+n*t)$ tuples in matrix A and matrix B, so there are $(m*n+n*t)$ records after mapping. In the shuffle procedure, all these records are sorted and sent to different reducers. In the Row Divided Method, it only divides the matrix A by row and doesn't divide matrix B. Thus in the shuffle procedure, it only needs

Table II
FEATURES OF ALL DATA SETS

	nodes	edges	diameter	Average degree
wiki-vote	7115	103689	7	14.6
P2P-Gnutella05	8846	31839	9	3.6
P2P-Gnutella31	62586	147892	11	2.4
Web-Stanford	281903	2,312,497	740	8.2
wiki-Talk	2,394,385	5,021,410	9	4

Table III
WIKI-VOTE DATASET

	Pre-process	1st iteration	2nd iteration	3rd iteration	4th iteration	Final Analysis
Initial	1min30s	3min26sec	1h39min4sec	1h57min42sec	2h2min14sec	-
STM	4min12sec	6min33sec	7min29sec	8min26sec	8min32sec	-
RDM	1min6sec	2min13sec	2min13sec	2min16sec	2min18sec	-

Table IV
P2P-GNUTELLA05 DATA SET

	Pre-process	1st iteration	2nd iteration	3rd iteration	4th iteration	Final Analysis
Initial	1min30s	1min6sec	1min8sec	2min15sec	8min14sec	-
STM	6min14sec	9min38sec	9min34sec	9min36sec	10min8sec	-
RDM	1min6sec	2min12sec	2min12sec	2min15sec	2min12sec	-

Table V
P2P-GNUTELLA31 DATASET

	Pre-process	1st iteration	2nd iteration	3rd iteration	Final Analysis
Initial	1min33 sec	1min36 sec	7min28 sec	1h23min11sec	-
STM	6min56 sec	9min28 sec	12min23 sec	12min	-
RDM	1min10sec	2min45sec	2min43sec	3min2sec	-

to sort m records, which is much less than that the Single Tuple Method will have to sort. Therefore, it reduces the expenses of sorting and communication greatly.

3) Comparison between two methods: The difference between the two methods mentioned before is the division granularity. Single Tuple Method is divided by element unit, and Row Divided Method is divided by row unit. As we expected, the first method should be more effective than the second one, because it has higher parallelism than the second one does. But the result of the experiments is totally opposite, which shocks us.

Then we analyze the amazing results. In the parallel Map-Reduce Computing Model, there exist data exchanging procedures between map procedure and reduce procedure, which are sort and shuffle procedures. In the sort procedure, all outputs of the map procedure are sorted. During the sorting, different nodes need to communicate with each other and thus communication cost are incurred. When sorting is finished, shuffle procedure sends the sorted results to different reducer. The time consuming in sorting cannot be ignored. High level parallel is important but the consuming of the sort and shuffle procedures cannot be neglected. The efficiency is determined by several factors, high parallelism is only one among them.

IV. THE EXPERIMENTAL RESULTS

This paper realizes all the algorithms of SimRank computation on a Hadoop platform with 20 slave machines and a master machine. Each machine has 6 cores, and installs Linux of Redhat Enterprise version. All the experiments are developed by JAVA language and using hadoop API. Three methods are tested on five different scale data sets, and the efficiencies of different methods are compared. The data set comes from Stanford data website [7].

A. Comparison between the several methods

The three different algorithms of SimRank computation are tested on five different scale data sets, and the differences among them are compared. Table II shows the features of each data sets.

Wiki-vote data set contains 7115 nodes and 103689 edges. Its details are showed in table III. P2P-Gnutella05 data set

Table VI
WEB-STANFORD DATASET

	Pre-process	1st iteration	2nd iteration	3rd iteration	Final Analysis
Initial	1min57sec	2h2min37sec	104h49min58sec	-	-
STM	3min20 sec	7min1sec	88min32sec	5h42min34sec	-
RDM	1min18sec	29min55sec	23min50sec	27min25sec	-

Table VII
WIKI-TALK DATASET

	Pre-process	1st iteration	2nd iteration	3rd iteration	4rd iteration	5th iteration	6th iteration
RDM	1min28sec	6h38min28s	5h47min54s	5h45min6sec	5h31min3sec	5h29min12	5h31min51s

contains 8846 nodes and 31839 edges, details are showed in table IV. P2P-Gnutella31 data set contains 62586 nodes and 147894 edges, details are showed in table V. Web-Stanford data set contains 281973 nodes and 2312497 edges. Since the graph is big, less iteration is carried out. Details are showed in table VI. Wiki-talk data set is too large that only RDM can give the final results, details showed in table VII.

The tables above show the result of three algorithms on different scale data sets. The efficiencies of different methods are determined by several factors, such as the number of nodes, the number of edges, the average degree of each node in the graph, and so on. For example, the number of nodes in P2P-Gnutella05 is larger than the number of nodes in wiki-vote, but the edges in P2P-Gnutella05 is less than the edges in wiki-vote, data set. For the Initial Iterative method ,the performance on the P2P-Gnutella01 data set is faster than that on the wiki-vote data set.

There are millions of nodes in the wiki-talk data set, except the RDM can deal with this kind scale of data set, others are not suitable for the data set of this scale. RDM has a better scalability.

V. APPLICATION IN SOCIAL RECOMMENDER SYSTEM

Social Recommender system plays an important role in many practical applications that help users to deal with information overload and provide personalized recommendations to them. The collaborative filtering recommendation is one of the most popular recommendation method. There are two categories of collaborative filtering methods. One is item-based and the other one is user-based. In Recommendation system, users usually mark item with score scaling from 0 to 5, which consists the scoring matrix. Collaborative recommendation methods using scoring matrix suffer from the problems such as cold start, sparsity, and so on. In order to resolve these problems, this paper proposes one recommendation method which is based on the linkage relationships among objects.

Although most of the work on collaborative filtering has focused on the traditional two-dimensional user-item matrix, there has been a recent increase of interest in integrating additional information to recommendations. This is probably due to the richness of additional information. This paper analyzes the similarity between objects with user-tag-item linkage. Tag System is a hot web system in Internet, in which users can issue any tags to items if the user consider the tag is suitable for the item. In this way, the tags can be looked as the content set of the item, the same as words to a document. So there is an tag-item relationship. As we know, the importance of different tag to the same item is not the same. In the Tag System, an user can label several tags to an item and the same tag may be issued to the same item by several users. Figure 1 shows an example of graph with the user-tag-item relationships.

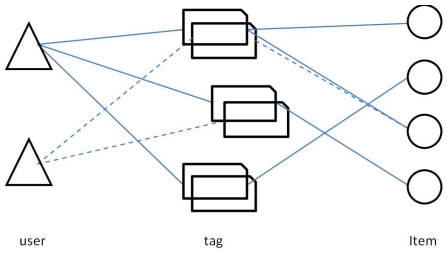

Figure 1. Graph with User-Tag-Item relationships

We can apply the previously discussed parallel SimRank computing method on this kind of graph to compute the similarities between the user and items, and then recommend the most similar items to the users.

VI. RELATED WORKS

Many methods have been proposed to better understanding of graph. Here we briefly describe the work that is most relevant to the current work. There is a lot of research work on static graph analysis, including power laws discovery [25], clustering and community identification [26], [15], frequent pattern mining [33], [32], and node ranking [27], [17].

There are two categories of methods to compute node similarity. The first one is based on text of items [14].

The second one is based on link relations [16], [18], [12], [19]. In research [24], the above two kinds of measures are evaluated and link-based measure produced systematically better correlation with human judgements than the former one.

SimRank [16] is a measure of the second category. Xi et al. proposed another node similarity computing algorithm called SimFusion [31] that utilizes the similar idea to compute node similarity scores. However, the time complexity of the initial SimRank or SimFision computation algorithms are very high.

There are many optimization techniques to reduce the computation cost of SimRank. Fingerprint-SimRank [13] pre-computes some steps of random walk path from each object. Although it improves computational performance of SimRank, Fingerprint-SimRank has highly cost of high space complexity. Lizorkin et al. [22] estimates the accuracy of computing SimRank and presents three optimization strategies to speed up the computation of SimRank. [5] proposes single-pair SimRank Computation algorithm. Overall, these methods are all based on one computer, none of which has considered to utilize the parallel framework of Map-Reduce. [3] proposes one parallel method to compute the SimRank. There are also some research works which use GPUs as a parallel hardware accelerator for SimRank computation [4]. [10] discussed the challenge and advances to implement sparse matrix multiplication under parallel architecture, and [9] used graphic processers to further improve the performance.

Map-Reduce was created by Google mainly to process big volume of unstructured data. Now it is widely used in several application, such as machine learning, data mining, text processing, and so on. Map-Reduce is a good tool for processing large scale graph. In the Map-Reduce computing model, the user only needs to provide a map function and a reduce function. The map function is applied to all input rows of the data set and produces an intermediate output that is aggregated by the reduce function later to produce the final result. Many more algorithms including graph processing algorithms [1] can be run on the Map-Reduce platform, and it can handle not only the unstructured data but also structured data efficiently [2]. This paper designs three new algorithms for SimRank computation based on Map-Reduce, including the parallel algorithm for the Initial Iterative Method and the algorithms for the Matrix Multiplication method.

Recently, there is also an increasing interest in mining dynamic graphs, such as group or community evolution [29], [8], power laws of dynamic graphs [20], dynamic tensor analysis [28], and dynamic clustering [11]. In terms of similarity updating, to the best of our knowledge, the only two existing papers are [30] and [21]. [30] proposed two algorithms to update the similarity matrix incrementally based on the Random Walk with Restart (RWR) model for a bipartite graph. [30] considers the incremental SimRank update problem on evolving graphs.

VII. Conclusion

This paper combines with the feature of MapReduce Computing Model ,designs several novel methods for SimRank computing, which not only improve the efficiency but also can deal with large scale data set. By comparing the several different methods ,we find that Monte Carlo Method is the fastest among all the parallel methods, Initial Formula Method is not that quick but it ensures the accuracy. Compared with the existing initial method, this parallel method is improved much in efficiency. Matrix multiplication method has good scalability especially the Row Divide method. This strategy can also be applied to other matrix algorithm.

Acknowledgment

This work was partially supported by NSFC under the grant No.61070056, 61033010, 61272137, 61202114, National Basic Research Program of China (973 Program)(No. 2012CB316205), and National Social Science Foundation of China (Project Number: 12&ZD220). It was partially done when the authors worked in SA Center for Big Data Research in Renmin University of China. This Center is funded by a Chinese National 111 Project Attracting.

References

[1] J. Lin, M. Schatz Design patterns for efficient graph algorithms in MapReduce. Eighth Workshop on Mining and Learning with Graphs 2010, pp.78-85.

[2] T. Kaldewey, E. Shekita, S. Tata Clydesdale: Structured Data Processing on MapReduce. EDBT 2012, pp.15-25.

[3] W. Yu, X. Lin, J. Le Taming Computational Complexity:Efficient and Parallel Simrank Optimizations on undirected Graphs. WAIM 2010,LNVS 6184,pp 280-296,2010

[4] G. He, H. Feng, C. Li, H. Chen Parallel simrank computation on large graphs with iterative aggregation. Proceedings of the 16th ACM SIGKDD 2010.

[5] P. Li, H. Liu, J. Yu, J. He, X. Du Fast Single-Pair SimRank Computation. SIAM International Conference on Data Mining ,2010,p.571-582

[6] J. Zhou, N. Bruno, M. Wu SCOPE: parallel databases meet MapReduce The VLDB Journal, published online 2012, DOI 10.1007/s00778-012-0280-z

[7] http://snap.stanford.edu/data/

[8] L. Backstrom, D. Huttenlocher, and J. Kleinberg. Group formation in large social networks: membership, growth, and evolution. In *Proc. of the 12th Int'l Conference on Knowledge discovery and data mining(KDD'06)*, 2006.

[9] N. Bell and M. Garland. Efficient sparse matrix-vector multiplication on cuda. In *Technical Report NVR-2008-004*, 2008.

[10] A. Buluç and J. R. Gilbert. Challenges and advances in parallel sparse matrix-matrix multiplication. In *ICPP '08: Proceedings of the 2008 37th International Conference on Parallel Processing*, pages 503–510, Washington, DC, USA, 2008. IEEE Computer Society.

[11] Y. Chi, X. Song, D. Zhou, K. Hino, and B. L. Tseng. Evolutionary spectral clustering by incorporating temporal smoothness. In *Proc. of the 13th Int'l Conference on Knowledge discovery and data mining(KDD'07)*, 2007.

[12] C. Faloutsos, K. S. McCurley, and A. Tomkins. Fast discovery of connection subgraphs. In *KDD '04: Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 118–127, New York, NY, USA, 2004. ACM.

[13] D. Fogaras and B. Racz. Scaling link-based similarity search. In *Proc. of the 14th Int'l Conference on World Wide Web (WWW'05)*, 2005.

[14] P. Ganesan, H. Garcia-Molina, and J. Widom. Exploiting hierarchical domain structure to compute similarity. *ACM Trans. Inf. Syst.*, 21(1):64–93, 2003.

[15] M. Girvan and M. Newman. Community structure in social and biological networks. In *Proc. Of the National Academy of Sciences*, 2002.

[16] G. Jeh and J. Widom. Simrank: a measure of structural-context similarity. In *KDD '02: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 538–543, New York, NY, USA, 2002. ACM.

[17] J. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM*, 1999.

[18] Y. Koren, S. C. North, and C. Volinsky. Measuring and extracting proximity in networks. In *KDD '06: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 245–255, New York, NY, USA, 2006. ACM.

[19] E. A. Leicht, P. Holme, and M. E. J. Newman. Vertex similarity in networks. *Physical Review E*, 73:026120, 2006.

[20] J. Leskovec, J. M. Kleinberg, and C. Faloutsos. Graphs over time: densification laws, shrinking diameters and possible explanations. In *Proc. of the 13th Int'l Conference on Knowledge discovery and data mining(KDD'07)*, 2007.

[21] C. Li, J. Han, G. He, X. Jin, Y. Sun, Y. Yu, and T. Wu. Fast computation of simrank for static and dynamic information networks. In *EDBT'10*, 2010.

[22] D. Lizorkin, P. Velikhov, M. Grinev, and D. Turdakov. Accuracy estimate and optimization techniques for simrank computation. In *Proc. of the 34st Int'l Conference on Very Large Databases (VLDB'08)*, 2008.

[23] Y. Zhang, Q. Gao, L. Gaoy, C. Wang iMapReduce: A Distributed Computing Framework for Iterative Computation datacloud'2011

[24] A. G. Maguitman, F. Menczer, F. Erdinc, H. Roinestad, and A. Vespignani. Algorithmic computation and approximation of semantic similarity. *World Wide Web*, 9(4):431–456, 2006.

[25] M.E.J.Newman. The structure and function of complex netwroks. *SIAM Review*, 2003.

[26] A. Ng, M. Jordan, and Y. Weiss. On spectral clustering: Analysis and an algorithm. In *Proc. Of the Advances in Neural Information Processing Systems(NIPS)*, 2002.

[27] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. *Technical report, Stanford University Database Group, http://citeseer.nj.nec.com/368196.html*, 1998.

[28] J. Sun, D. Tao, and C. Faloutsos. Beyond streams and graphs: dynamic tensor analysis. In *Proc. of the 12th Int'l Conference on Knowledge discovery and data mining(KDD'06)*, 2006.

[29] C. Tantipathananandh, T. Y. Berger-Wolf, and D. Kempe. A framework for community identification in dynamic social networks. In *Proc. of the 13th Int'l Conference on Knowledge discovery and data mining(KDD'07)*, 2007.

[30] H. Tong, S. Papadimitriou, P. S. Yu, and C. Faloutsos. Proximity tracking on time-evolving bipartite graphs. In *Proc. of SDM*, 2008.

[31] W. Xi, E. A. Fox, W. Fan, B. Zhang, Z. Chen, J. Yan, and D. Zhuang. Simfusion: measuring similarity using unified relationship matrix. In *SIGIR '05: Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 130–137, New York, NY, USA, 2005. ACM.

[32] X. Yan and J. Han. Closegraph: Mining closed frequent graph patterns. In *Proc. of the 9th Int'l Conference on Knowledge discovery and data mining(KDD'03)*, 2003.

[33] X. Yan, P. S. Yu, and J. Han. Substructure similarity search in graph databases. In *Proc. Of ACM-SIGMOD Int'l Conference on Management of Data*, 2005.

[34] J. Dean, and S. Ghemawat MapReduce: simplified data processing on large clusters. in proceedings of the 6th conference on Symposium on Opearting Systems Design and Implementation ,2004.

[35] Y. Bu, B. Howe, M. Balazinska, and M. Ernst HaDoop :Efficient Iterative Data Processing on Large Clusters. Proceedings of the VLDB Endownment, Vol3, No.1.

[36] C. Olston, B. Reed U. Srivastava Pig Latin: A Not-So-Foreign Language for Data Processing Proceedings of SIGMOD'08, 1099-1111