Parallel Mining Frequent Patterns over Big Transactional Data in Extended MapReduce

Hui Chen*, Tsau Young Lin†, Zhibing Zhang* and Jie Zhong*
*School of Software and Communication Engineering
Jiangxi University of Finance and Economics, Nanchang, China
Email: flychcn@gmail.com, zhbzhang@163.com, zjhadoop@gmail.com
†Department of Computer Science
San Jose State University, San Jose, CA, USA
Email: tylin@cs.sjsu.edu

***Abstract*—In big data era, data size has raised from TB-level to PB-level. Traditional algorithm can not satisfy the needs of big data computing. This paper design a parallel algorithm for mining frequent pattern over big transactional data based on an extended MapReduce Frame. In which, the mass data file is firstly split into many data subfiles, the patterns in each subfile can be quickly located based on bitmap computation by scanning the data only once. And the computing results of all subfiles are merged for mining the frequent patterns in the whole big data. In order to improve the performance of the proposed method, the insignificant patterns are pruned by a statistic analysis method when the data subfiles are processed. The experimental results show that the method is efficient, strong in scalability, and can be used to efficiently mine frequent patterns in big data.**

***Index Terms*—Big Data; Frequent Pattern Mining; Bitmap Computation; MapReduce;**

I. Introduction

The amount of data in our world has been exploding, companies capture trillions of bytes of information about their customers, suppliers and operations. Millions of networked sensors are being embedded in the physical devices such as mobile phones and automobiles, sensing, creating, and communicating data. Multimedia and individuals with smartphones and on social network sites will continue to fuel exponential growth. Big data [1], [2], [3] is now part of every sector and function of the global economy [4]. Big data, a collection of data sets, is so large and complex that it is beyond the ability of typical database software tools to capture, store, manage, and process the data within a tolerable elapsed time.

According to the report from the McKinsey Global Institute [4], 15 out of 17 business sectors in the United States have more data stored per company than the U.S. Library of Congress. Wal-Mart is a good example: The retail giant handles more than one million customer transactions very hour, importing this data into databases estimated to contain more than 2.5 petabytes of data which is equivalent to 167 times the information contained in all the books in the Library of Congress. And the report by Intel [5] shows that 82 percents business transactions in database need to be analyzed. One of the interesting works of mining retail transactional data is to find the frequent patterns.

During the past two decades, many works [6], [7], [8], [9], [10], [11], [12], [13] on mining frequent patterns of transactional database have been proposed, and the most famous methods are Apriori-like and Pattern-growth algorithms. The former is first proposed by Agrawal and Srikant [8], which is based on an interesting downward closure property, called Apriori, among frequent k-itemsets: *A k-itemset is frequent only if all of its sub-itemsets are frequent.* The Apriori heuristic can help to find interesting patterns from transactional database. But an apriori-like algorithm may suffer from nontrivial costs, because it needs repeatedly scan the database and check a huge size of candidates by pattern checking. The representative work of the Pattern-Growth algorithm, named FP-growth, is presented by Han et al. [7]. FP-growth works in a divide-and-conquer way, and transforms the problem of finding long frequent patterns to searching for shorter ones recursively and then concatenating the suffix. An FP-growth method can find the frequent patterns without candidate generation by scanning the database only twice.

However, the existing Apriori-like and Pattern-growth algorithms are not suitable to mine a big data, for the size of a big data is so big that it is inefficient to scan the data multiple time. Moreover, when processing a big data, it requires massively parallel software running on hundreds or even thousands of cheap computers that act as a cluster, instead of autonomously algorithm running on one expensive supercomputer. So it is necessary to find a paralleled algorithm [14] that can be run on massive computers to mine the frequent patterns in a big transactional data, but it is regretful that there is no such a algorithm.

In this paper, we propose a parallel algorithm running in MapReduce to mine the frequent patterns over big transactional data. In which, the data are firstly split into

Corresponding author: Tsau Young Lin

This work was partly supported by the National Science Foundation of China under grant NO.60763002, 61262033; the Natural Science Foundation of Jiangxi Province, China under grant NO.2008GZS0021; the Educational Commission of Jiangxi Province, China under grant No.GJJ13303; and the Middle Age and Young Teachers Development Program of Undergraduate Universities in Jiangxi Province, China.

978-1-4799-1282-7/13/$31.00 ©2013 IEEE

many data subsets for synchronously processing on hundreds or thousands computers. Additionally, a probability model is designed to prune insignificant patterns for better performance. Finally, the frequent patterns in the whole big data can be quickly selected from the significant patterns. The primary contributions of this paper are:

- Our method is novel. So far as I known, there is no analogous algorithm that is designed to mine frequent patterns in big transactional data.
- The transactional data are scanned only once and the patterns in data files can be quickly located by bitmap computation under limited memory usage.
- A probability model is designed to pruned the insignificant patterns when processing data subfile, which can greatly cut down the processing time and size of intermediate data. At the same time, the correctness of mining results can also be guaranteed.

The rest of this paper is organized as follows. Section II presents related work and simply introduces the MapReduce framework. Section III discusses the method of mining frequent patterns using bitmap computation. Section IV extends the MapReduce frame and proposes the method of parallel mining frequent patterns in big transactional data. Section V explains the experimental results. Section VI concludes this work.

II. Background

A. Related Work

Frequent pattern mining was first proposed by Agrawal et al. [8] for market basket analysis in the form of association rule mining. It analysis customer buying habits by finding associations between the different items that customers place in their "shopping baskets". The Apriori method proposed by Agrawal et al. is based on an anti-monotonic Apriori heuristic: if any length k pattern is not frequent in the database, its length (k+1) super-pattern can never be frequent. The essential idea is to iteratively generate the set of candidate patterns of length (k+1) from the set of frequent patterns of length k (for $k \geq 1$), and check their corresponding occurrence frequencies in the database [7]. In order to improve the performance of Apriori-like method, some works [9], [10], [11] are proposed to reduce the number of generated candidates. Even though, an Apriori-like algorithm may still be tedious to repeatedly scan the database and check a large set of candidates by pattern matching in situations with prolific frequent patterns, long patterns, or quite low minimum support thresholds.

In order to avoid generating a huge set of candidates, Han et al. [7] proposed an FP-growth method to mine transactional transactions without candidate generation. An FP-growth method can achieve much better performance than an Apriori method by scanning database only twice. At the first time of scanning database, all frequent data items are selected and sorted in a frequency descendant order, and then an FP-tree can be built to maintain the projections of the transactions that only contain frequent items at the second time of scanning database. At last, the frequent patterns can be quickly selected from FP-tree by applying an FP-growth algorithm.

Both Apriori-like methods and FP-growth method can not be directly applied to mine a big transactional data because the following reasons: for a Apriori-like method, it firstly is rather costly to generate numerous candidates, and additionally it also is impractical to repeatedly scan the huge data to check whether a pattern is frequent or not. And for FP-growth method, it also needs to scan the data twice, and it is also expensive to perform projection and sort operations on each transaction.

B. MapReduce frame

Hadoop MapReduce [15], [16] is the most popular open source implementation of the MapReduce framework proposed by Google. Generally speaking, a Hadoop MapReduce job mainly consists of three user-defined functions: *map*, *combine* and *reduce* [17]. The input of a Hadoop MapReduce job is a set of key-value pairs (k, v) and the map function is called for each of these pairs. The map function produces zero or more intermediate key-value pairs (k', v'). Then, the Hadoop MapReduce framework groups these intermediate key-value pairs by intermediate key k' and calls the reduce function for each group. Finally, the reduce function produces zero or more aggregated results. In order reduce the intermediate data sent to the reducer, an optional combine function can perform a local consolidation.

III. Frequent pattern mining based on bitmap Computation

Given a scheme of relational database R and its attribute set $A = \{A_1, A_2, \cdots, A_n\}$, where the range of attribute A_i is D_i ($1 \leq i \leq n$), then the universal set of the entities in R will be $U = D_1 \times D_2 \times \cdots \times D_n$ ($1 \leq i \leq n$). Suppose that r is a relation of R, then the real range of attribute A_i in r will be $\overline{D_i}$ and $\overline{D_i} \subset D_i$, and it is called the active domain of r on A_i and formally define it as follow.

Definition 1. *Suppose that r is a relation of scheme R, the real rang of attribute A_i in r ($1 \leq i \leq n$) is d_i ($d_i \subset D_i$). Then d_i is named the **active domain** of r on A_i, and denoted by $ADom(A_i, r) = \{d \in D_i | \exists t \in r \text{ and } t(A_i) = d\}$. With no confusion, we simple $ADom(A_i, r)$ to be $ADom(A_i)$.*

Definition 2. *In a relation r, it implicates there exists a mapping, F, from $ADom(A_i)$ to r, that is $F : V \rightarrow ADom(A_i)$, where V is the tuple set in r. For any value $c \in ADom(A_i)$, the set of tuples whose values on attribute A_i being c can be denoted $F^{-1}(c) = \{t | t \in V, t(A_i) = c\}$, and named the **inverse image** of c in r.*

Definition 3. *Suppose that there are m tuples in relation r, c is a value of an attribute A_i, then a m-bit binary number can be used to dedicate the tuples that contain c, the m-bit binary number is called the **bitmap** of value c*

in r on A_i, and denoted $Bin(A_i, c)$. If $t_s \in F^{-1}(c)$, then $Bin_s(A_i, c) = 1$ or else $Bin_s(A_i, c) = 0$, where t_s is the s^{th} tuple in r, $F^{-1}(c)$ is the inverse , $Bin_s(A_i, c)$ is the s^{th} bit in $Bin(A_i, c)$.

Definition 4. *Suppose that $(c_1, c_2, \cdots, c_l)$ is a component in relation r on attribute set $(A_1, A_2, \cdots, A_l)$, then $(c_1, c_2, \cdots, c_l)$ is thought as a pattern in r, denoted by P. The number of tuples that occur P is the* **frequency** *of P, denoted $freq(P)$. The support of P, denoted by $sup(P)$, equals $freq(P)/N$, where N is the size of r. Given user support threshold θ, P is frequent if $sup(P) \geq \theta$, or else it is infrequent.*

Suppose that r is a relation of R, and $A_1, A_2, \cdots, A_m$ are the m attributes. For any a component $(c_1, c_2, \cdots, c_l)$ $(1 \leq i \leq l)$, its bitmap in r on $(A_1, A_2, \cdots, A_l)$, denoted by $Bin((c_1, c_2, \cdots, c_l), (A_1, A_2, \cdots, A_l))$, represents the tuples in which the components on attributes $A_1, A_2, \cdots, A_l$ equals $c_1, c_2, \cdots, c_l$ respectively. Obviously there is $Bin((c_1, c_2, \cdots, c_l), (A_1, A_2, \cdots, A_l)) = \&_{i=1}^{l} Bin(A_i, c_i)$, where $\&$ is the bitwise-and operator. The component $(c_1, c_2, \cdots, c_l)$, denoted by P, can be thought to be a pattern of length being l in r, and the support of P, denoted by $sup(P)$, equals the ratio of the bites being '1' in $Bin((c_1, c_2, \cdots, c_l), (A_1, A_2, \cdots, A_l))$. Given a user support threshold θ $(0 < \theta < 1)$, P is frequent if $sup(P) \geq \theta$.

IV. Analysis on Parallel mining frequent patterns in big data

In this section, we will firstly extend the traditional MapReduce frame, and then propose the method of parallel mining the frequent patterns in a big transactional data based on the extended frame.

A. Extended MapReduce frame

The MapReduce frame provides an efficient solution to process the big data in a divide-and-conquer way. However, the original MapReduce frame is designed for search engine, and it is not efficient if the MapReduce frame is directly used to mine frequent patterns in big data. In order to efficiently mine the frequent patterns in a big transactional data using a MapReduce frame, the original MapReduce frame is extended as shown in Figure 1, in which, serval improvements are made: (1) After the big data is split into small size sub-files, the bitmaps of data items in each sub-file are computed, and then the insignificant patterns can be quickly located by *SPatMiner* function with limited memory usage. (2) The insignificant patterns can be safely pruned based on probability analysis, as discussed in Section IV-B, which can greatly cut down the size of intermediate computing results and the processing time spent on each sub-file. (3) After all the significant patterns in all data sub-files are mapped as $< key, value >$ pairs and merged into output file, the frequent patterns cab be quickly selected by *FPatMiner* function.

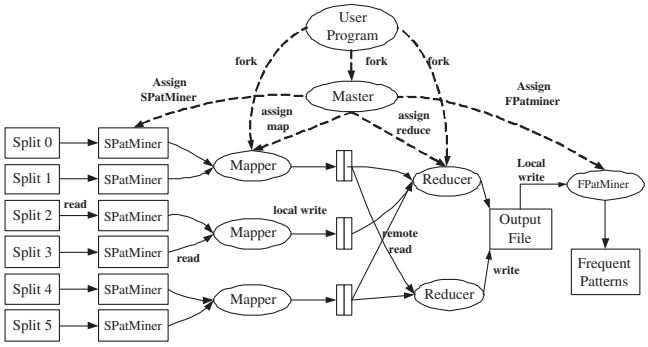

Fig. 1. Extended MapReduce framework for mining big transactional data

B. Parallel mining frequent pattern in a big data

Suppose that a big data, denoted by $BData$, contains n data items $\{A_1, A_2, \cdots, A_n\}$ and has N transactions $\{T_1, T_2, \cdots, T_N\}$. $BData$ can be thought as a relation with n attributes and N tuples, in which each attribute has two possible values '0' and '1'. That is to say that the active domain of each attribute is $('0', '1')$. Transaction T_i is thought as the i^{th} tuples in the relation. If the component of T_i on an attribute A_i equals '1' then T_i contains item A_i, or else it does not. When mining the frequent patterns in $BData$, the bitmap computation method can be used to efficiently find the patterns and compute their frequencies.

When mining a big data containing N transactions with the extended MapReduce frame, the big data is first divided into m data subsets each has $n(= N/m)$ transactions. Suppose that P is a pattern in the big data, and P occurs x times in one of the data subsets, then x ranges from zero to n. Suppose that the probability that the number of data subsets in which P occurs x times is subject to a probability density function (**PDF** in short) $f(x)$, $x = 0, 1, 2, \cdots, n$. Obviously there are $m \times f(x)$ data subsets in which P occurs x times, and the frequency of P can be obtained by following equation.

$$freq(P) = \sum_{x=0}^{n} x \times m \times f(x) \tag{1}$$

Given the user support threshold θ $(0 < \theta < 1)$, P is frequent if $freq(P) \geq \theta N$.

As has mentioned, in order to achieve better performance, it is important to prune the insignificant patterns (as formally defined in definition 5) when processing a data subset. Next, we will discuss the method that can guarantee the correctness of mining results even though the insignificant patterns are pruned as they are first found.

Definition 5. *For any pattern P in a data subset of the big data, if $sup(P) < \xi$, then P is thought as* ***insignificant pattern*** *where ξ is the frequency threshold of insignificant pattern and it will determined in the following context.*

Suppose that P is a patterns in a big data, if P in some

data subsets are pruned when processing the data subsets because its frequency is less than ξ, then the frequency of P in whole big data might be less than their true frequencies. The frequency of P under this situation is named its *reduced frequency* and denoted by $freq'(P)$. Obviously, $freq'(P)$ can be calculated by following equation.

$$freq'(P) = \sum_{x=\xi}^{n} x \times m \times f(x) \tag{2}$$

For clearly, in the following text, the *true frequency* of P, denoted $freq(P)$, means its frequency without pruning. Now suppose that P happens to be a pattern whose support equals θ in whole big data, then it is obvious that $freq'(P) \leq freq(P)$ because P will be pruned in some data subsets. However we still want to select P as a frequent pattern because its real support equals θ. In order to realize this objective, P is regarded as a frequent pattern if $freq'(P) \geq (\theta - \epsilon) \times N$, where ϵ $(0 < \epsilon < \theta)$ is the permissible error parameter and $\epsilon \times N$ is maximal error between the true frequency and the reduced one of P. Then we can have the following inequality.

$$freq(P) - freq'(P) \leq \epsilon \times N \tag{3}$$

According to the above analysis, the inequality 3 can be simplified to be as follows.

$$\sum_{x=0}^{\xi} xf(x) \leq \epsilon n \tag{4}$$

It is hard to derive a formal expression for ξ from above Inequality 4 for the **PDF** $f(x)$ is unknown. However, give a big data, the **PDF** $f(x)$ of a pattern P is ascertained. so it is perfectly sure and easy to calculate the maximal of ξ after parameters n and ϵ are provided. As known to all, there might be many patterns whose support being θ in a big data, and their **PDF**s also might be different. It is very costly to analyze the **PDF**s of such patterns when mining a big data. A straightforward method is to choose an appreciate **PDF** to represent the others. In the next subsection, we will discuss how to choose a appreciate **PDF** in details.

C. Analysis on probability density function

Suppose that $PS = \{P_1, P_2, \cdots, P_i, \cdots, P_s\}$ are s patterns whose supports are right to be θ, and their **PDF**s are $f_1(x), f_2(x), \cdots, f_i(x), \cdots, f_s(x)$ respectively. Next, we will discuss how to choose an appreciate **PDF** with the correctness requirement based on Probability Analysis Theory. For the purpose of clearly explaining our method, the following definition is first presented, and then serval propositions are derived.

Definition 6. *Given a length k pattern P, and its length $l(0 < l < k)$ sub-pattern, denoted by P' ($P' \subset P$), P is called one of the **supper-patterns** of P', and P' is called one of the **sub-patterns** of P.*

Then according to Apriori Theory, for any a pattern P in a big transactional data, (1) if P is frequent, then all its sub-patterns must be frequent too; (2) if P is infrequent, then all its supper-patterns must be infrequent too. If all patterns whose frequencies being θN in a big data are mined to be frequent, then their sub-patterns must be frequent too. Similarly, if all patterns whose frequencies being $\theta N - 1$ are mined to be infrequent, then their supper-patterns must be infrequent too.

Now, let consider the s patterns in PS whose supports are θ. According to Inequality 4, s values for ξ, $\xi_1, \xi_2, \cdots, \xi_s$, can be obtained respectively, and suppose that $\xi_1 \leq \xi_2 \leq \cdots \leq \xi_s$.

Proposition 1. *If the minimum value of ξ, denoted ξ_{min}, is chosen to prune insignificant patterns, then all patterns in PS must be frequent.*

Proof: If $\xi_{min} = \xi_1$ is used to prune the insignificant patterns in each data subset, then P_1 is frequent because $freq'(P_1) = \sum_{x=\xi_1}^{n} x \times m \times f_1(x) \geq (\theta - \epsilon)N$ according to the Equation 2. For any pattern $P_i \in PS$ $(1 < i \leq s)$ must be frequent too because there is

$$freq'(P_i) = \sum_{x=\xi_1}^{n} x \times m \times f_i(x) \geq \sum_{x=\xi_i}^{n} x \times m \times f_i(x) \tag{5}$$

Because P_i is an arbitrary pattern in PS, so all patterns in PS must be frequent. ■

To measure the quality of the frequent pattern mining results, two performance metrics [12], namely **recall** and **precision**, are commonly used. Given a set of true frequent patterns X and a set of retrieved frequent patterns Y, recall is $\frac{|X \cap Y|}{|X|}$ and precision is $\frac{|X \cap Y|}{|Y|}$. Based on Proposition 1, if ξ_{min} is used to prune insignificant patterns, all patterns in PS and their sub-patterns are sure to be frequent. Then the recall of the mining result will arrive at 100%.

Proposition 2. *If the maximum value of ξ, denoted ξ_{max}, is chosen to prune insignificant patterns, then all patterns in PS except P_s can not be frequent.*

Proof: If $\xi_{max} = \xi_s$ is used to prune the insignificant patterns in each data subset, then all patterns in PS except P_s are infrequent according to the analogous process of ratiocination as that in Proposition 1. ■

Based on Proposition 2 and above analysis, if ξ_{max} is set to be a little bigger that ξ_s, then all patterns in PS will be infrequent, and precision of the mining result will reach 100%.

When mining a real big data of huge size, it is also impractical to pre-analysis all the **PDF**s of the patterns with supports being θ for achieving 100% correctness, because there might be a mass of frequent patterns whose supports are θ in the big data. It is realizable to empirically choose two or more patterns whose supports are or near θ, approximatively analyze their PDFs, calculate the values

of ξ, and get the average of those ξs. As discussed in the correctness experiment in Section V-C, the recall and precision are acceptable if $\xi_{min} < \xi < \xi_{max}$.

D. Parallel algorithm implement

According to the parallel mining model discussed above, the parallel algorithm of mining frequent patterns in a big data based extended MapReduce is proposed. The process of the algorithm can be described as follow: (1) empirically choose two or more patterns whose supports are or near θ and get an appreciate value for ξ. (2) Split the big data into a series of dat subsets size of n. (3) Synchronously get the bitmaps of all data items in a subset, and obtain the significant patterns by pruning the patterns whose frequencies are less than ξ. (3) The significant patterns are mapped into $< key, value >$ pairs and then merged into a output file. (4) The approximate set of frequent patterns can be obtained by mining the output file.

The function SPatMiner() is designed to get the significant patterns in a data subset by applying bitmap computation, and prunes the insignificant ones according to the probability model discussed in Section IV-B. As shown in lines 1-6, the set of significant items in the processed data subset, denoted by SIS, can be obtained. By applying the pattern growth method, the bitmaps of the patterns combined by the significant items can be quickly calculated. At the same time, the significant patterns can be efficiently differentiated from the insignificant ones by checking the number of '1' in their bitmaps, which is shown in lines 9-23. Additionally, as shown in lines 18-22, if a pattern is insignificant, it will not grow any more, which can greatly decrease the number of candidate patterns and make the method efficient.

V. Simulation

In this section we report the performance of our extended Mapreduce running on a large cluster of machines study. The experimental setup is described in Section V-A, and the results are given in Section V-B.

A. Experimental setup

All of the programs were executed on a cluster consisted of 100 machines. Each machine had an Intel Core i3-2100 3.1 GHZ processor with Hyper-Threading enabled, 4GB of main memory, one 500GB IDE disks, and a gigabit Ethernet link. All of the machines were in the same hosting facility and therefore the round-trip time between any pair of machines was less than a millisecond. All programs were written in Java, and running on Hadoop 0.20.0 frame.

The synchronous data sets, generated by IBM synthetic data generator (http://www.almaden.ibm.com) using 20k distinction items and 128 millon transactions, were used in the simulation. In the experiment, the data file was firstly split into subfiles consists of l transactions. In order to evaluate performance, the following criteria were used in the experiments. (1) PT: The processing time of mining frequent patterns using extended MapReduce. (2) Recall: Recall measures the degree to which the system delivers all frequent patterns. (3) Precision: Precision measures the degree to which the system delivers only frequent patterns.

B. Performance results

In the experiments, l is set to be 128K, and each the data set is split into 1024 subfiles. The memory usage of computing the bitmaps of items in a data subfiles will be fixed to $128K \times 20K\%8bits = 320MB$. So the maximum memory usage will no more than 500M by taking other memory usage into account.

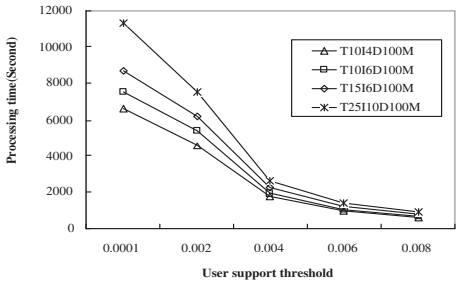

Fig. 2. The processing time on varying support threshold

Firstly, a series of experiments on datasets T10I4D128M, T10I6D128M, T15I6D128M and T25I10D128M are conducted to study the basic performance and the scalability of the proposed method by varying the user support threshold, θ, from 0.001 to 0.008. In the experiments, ϵ is fixed to be $0.1 \times \theta$, and ξ is set to be the maximum computed by Equation 4 based on the average probability density of the patterns whose supports are θ. When θ increases, more patterns are contained in the data set, not only more time is needed to process the patterns, but also a longer time is needed to transfer the intermediate results. Therefore, as shown in Figure 2, the processing time of proposed method increases quickly as θ is increasing. Additionally, as the average transaction length (T) and average pattern length (I) increase, there contain more patterns in the dataset and more time is needed to process more patterns and transfer intermediate computing results. But the increasing degree of processing time is limited even though the density of dataset increases greatly.

Secondly, the dataset T10I4D128M was used to study the performance of the proposed method on varying ϵ and ξ. Given a maximum support error ϵ, a maximum value for ξ, denoted ξ^*, can be calculated by Equation 4 based on the average probability density of the patterns whose supports are right to be θ. In the experiments, ϵ varies from 0.1 to 0.4 times of θ. and for each ϵ. Given a ϵ, a maximum value for ξ, denoted ξ^*, can be calculated, and ξ varies from 0.2 to 1 times ξ^*. In the proposed method, parameters ϵ and ξ are used to prune the insignificant patterns in each data subfile, which can greatly improve the performance

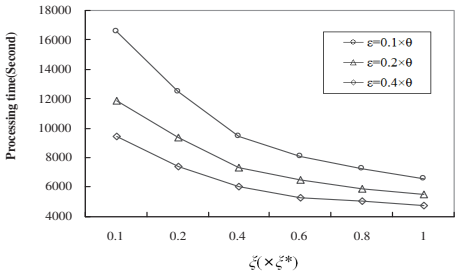

Fig. 3. The processing time on varying ϵ and ξ

of mining a data subfile. So bigger the value of ϵ and ξ are, more insignificant patterns will be pruned. Thus few patterns need to be saved and used to mine the frequent patterns in the whole data when processing a data subfile, and hence achieve better performance. So as shown in Figure 3, the processing time of the method decreases as ϵ and ξ increase.

C. Correction analysis

As have discussed in Section IV, in order to achieve better performance, some insignificant patterns will be pruned when processing the data subfile. As a result, some error will be brought into the mining result when mining the frequent patterns in whole data, because the information of some insignificant patterns are lost. In the experiments, we study the correctness of the proposed method by conducting a series experiments on T10I4D128M. We first analyze the PDFs of all 278 patterns whose supports are right to be θ in the dataset, then the minimum and maximum of ξ, denoted ξ_{min} and ξ_{max} respectively, can be calculated, and $\xi_{min} = 0.0000469$ and $\xi_{max} = 0.0000547$. As shown in Figure 4, the mining result can reach 100% recall if ξ_{min} is use to prune the insignificant patterns, on the contrary, the mining result will arrive at 100% precision if ξ_{max} is used.

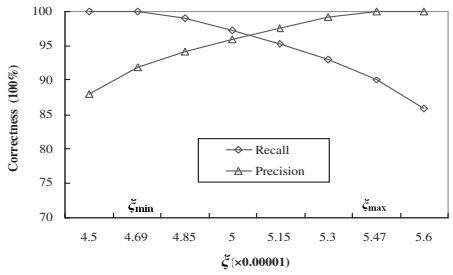

Fig. 4. The correction of mining result on varying ξ.

VI. Conclusion

By extending the MapReduce frame, a parallel algorithm is presented for mining the frequent patterns in a big transactional data. According to MapReduce frame, the big data is firstly split into a lot of data subfiles, and the subfiles are concurrently processed by a cluster consisting of hundreds or thousands computers. In order to improve the performance of proposed method, the insignificant patterns in the data subfile are efficiently located and pruned by probability analysis. The simulation result proves that the method is efficient and scalable, and can be used to efficiently mine frequent patterns in big data.

References

[1] A. Labrinidis and H. V. Jagadish, "Challenges and opportunities with big data," in *Proceedings of the VLDB Endowment*. VLDB, August 2012, pp. 2032–2033.

[2] W. Fan and A. Bifet, "Mining big data: current status, and forecast to the future," *ACM SIGKDD Explorations Newsletter*, vol. 14, no. 2, pp. 1–5, December 2012.

[3] G. Mishne, J. Dalton, Z. Li, and etc., "Fast data in the era of big data: Twitter's real-time related query suggestion architecture," in *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*. ACM, June 2013, pp. 1147–1158.

[4] "Big data: the next frontier for innovation, competition, and productivity," *McKinsey Global Insititute, http://www.mckinsey.com /insights/mgi/research/technology_and_innovation/big_data_the_next_frontier_for_innovation*, June 2011.

[5] "Intel peer research: Big data analysis, intel's it manager survey on how organizations are using the big data," *http://www.intel.com/content/www/us/en/big-data/data-insights-peer-research-report.html*, Auguest 2012.

[6] H. Chen, L. Shu, J. Xia, and etc., "Mining frequent patterns in a varying-size sliding window of online transactional data streams," *Information Sciences*, vol. 215, pp. 15–36, December 2012.

[7] J. Han, J. Pei, and Y. Yin, "Mining frequent patterns without candidate generation," in *Proceedings of the 9th international conference on Parallel Computing Technologies*, September 2007, pp. 623–631.

[8] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules in large databases," in *Proceedings of 20th International Conference on Very Large Data Bases*, September 1994, pp. 487–499.

[9] A. Savasere, E. Omiecinski, and S. B. Navathe, "An effective algorithm for mining association rules in large databases," in *Proceedings of the 21th International Conference on Vary Large Data Bases*, September 1995, pp. 432–444.

[10] F. Geerts, B. Gogethals, and J. V. D. Bussche, "A tight upper bound on the number of candidate patterns," in *Proceedings of IEEE International Conference on Data Mining*, November-December 2001, pp. 155–162.

[11] J. H. Chang and W. S. Lee, "Finding recent frequent itemsets adaptively over online data streams," in *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Augest 2003, pp. 487–492.

[12] J. X. Yu, Z. Chong, H. Lu, and etc., "A false negative approach to mining frequent itemsets from high speed transactional data streams," *Information Sciences*, vol. 176, pp. 1986–2015, July 2006.

[13] H. Chen, "Mining top-k frequent itemsets over data streams sliding window," *Accepted to be published in Journal of Intelligent Information Systems, DOI: 10.1007/s108844-013-0265-4*.

[14] R. K. Menon, G. P. Bhat, and M. C. Schatz, "Rapid parallel genome indexing with mapreduce," in *Proceedings of the Second International Workshop on MapReduce and its Applications*, June 2010, pp. 51–58.

[15] K. Shim, "Mapreduce algorithms for big data analysis," in *Proceedings of the VLDB Endowment*. VLDB, Auguest 2012, pp. 2016–2017.

[16] J. Dean and S. Ghemawat, "Mapreduce: A flexible data processing tool," *Comunications of the ACM*, vol. 53, no. 1, pp. 72–77, January 2010.

[17] J. Dittrich and J.-A. Quiané-Ruiz, "Efficient big data processing in hadoop mapreduce," *Proceedings of the VLDB Endowment*, vol. 5, no. 12, pp. 2014–2015, Auguest 2012.