

Identification Of Seed Users Via Short Messages Based On Hadoop

Ye Zhiwei

School of Software Engineering
South China University of Technology
Guangzhou, China
316553421@qq.com

Zhang Pingjian

School of Software Engineering
South China University of Technology
Guangzhou, China
pjzhang@126.com
* Corresponding author

Abstract—With the rapid growth of the text processing technology, many knowledge discovery approaches have been introduced to handle large corpus. Data mining methods such as clustering and categorization, for example, have found wide applications in corpus processing. Recently, association rule mining methods also have a place in this field. However, due to the huge amount of "items" contained in corpus, the traditional association rule mining algorithms encounter great effectiveness and efficiency challenges. In this paper, a new parallel association rule mining algorithm especially customized for corpus is developed and implemented using the MPI programming interface. The main ideas are to adopt a distributed inverted index hash table, and to design a communication scheme based on "chessboard decomposition" to accelerate the generation of candidate itemsets. Experiments are devised and conducted on the Tianhe-II Supercomputer of Guangzhou National Super Computing Center. The experimental results demonstrate that the new algorithm has achieved desirable performance, with a speedup rate of 16 when using 49 processes altogether.

Keywords—Corpus processing; parallel association rule mining; inverted index hash table; MPI; speedup

I. INTRODUCTION

Human society is experiencing the age of information explosion, data from various fields is accumulated at an amazing rate. Internet in China, for example, has become a huge warehouse of data [1]. Web pages are mainly composed of text, multimedia information and hyper-links among them, with text being the largest part. Since text is usually semi-structured or unstructured, how to deal with huge amounts of unstructured data presents a major challenge in the field of data mining and becomes a hot topic, attracting much attentions from either academic or industrial community. Most research work in corpus processing utilizes some sort of machine learning approaches. Graph-based methods, for example, has been used for natural language processing [2]. The Huffman encoding algorithm finds its role in improving accuracy of text clustering [3]. Methods for mining structural information from text corpus have been surveyed in [4]. Concept based methods such as formal concept analysis also contribute to knowledge processing of corpus [5,6].

Recently, association rules begin to play a role in this field [7,8,9]. A content-based book recommendation system has been developed by mining associations among Amazon books[10]. In [11], retrieval of clinical text has been improved using semantic-based association rule mining. However, the most prominent characteristics of text document is that it might contains a large amount of words, giving rise to huge number of frequent itemsets. Thus, traditional algorithms such as Apriori and FP-Growth could not scale well on text corpus.

Nowadays, high performance multi-core computer clusters together with supercomputers begin to provide powerful computing service for research institutes, organizations and enterprises as well. Parallelization solutions are more and more popular and become the trend of the big data era. As early as in 1996, Agrawal has proposed a parallel Apriori based association rule mining algorithm [12], which was then improved by various researchers. Nevertheless, due to the sparsity of terms in corpus and hence much fewer shared prefixes, FP-Growth algorithm tends to generate an FP-Tree with a large number of nodes and branches, which would greatly impact the efficiency. To overcome this difficulty, an HI-Apriori algorithm has been presented in [13], which adopts an inverted index hash table instead of an FP-Tree to hold the corpus. On one hand, HI-Apriori retains the advantage of FP-Growth, avoiding generating frequent itemsets and scanning the dataset multiple times. On the other hand, it is much efficient than FP-Growth when handling corpus.

In this paper, the HI-Apriori algorithm is extended to a parallelization version, called PHI-Apriori. First, the HI-Apriori algorithm is analyzed to identify the computational extensive area. Then, parallelization are designed w.r.t the hot spot, using the "chessboard decomposition" strategy. The PHI-Apriori algorithm is implemented via the MPI interface and finally, experiments are carried out in the Guangzhou National Super Computing Center. Numerical results show that PHI-Apriori achieves satisfactory speedup and scales well for corpus of various size and computing clusters of various configurations.

II. ANALYSIS OF HI-APRIORI

The main idea of HI-Apriori is to introduce an inverted index hash table which facilitates generation of frequent itemsets.

A. Preprocess of the corpus

Mining text usually requires some sort of preprocessing so that the remaining text only contains necessary information. Web pages crawled from the internet, for example, need to get rid of the tags, hyperlinks, multimedia, advertisements etc, with only pure text left. For some languages like Chinese, an additional step word segmentation must be carried out to tokenize the sentences. The HI-Apriori resorts to third-party libraries for this job and focuses on generation of frequent itemsets. Moreover, modal particle, as well as stop words, are filtered, leaving only meaningful terms.

B. Inverted index hash table

Inverted index hash table is an important tool in document processing, and plays an essential role in the field of information retrieval. In [13], it is found that inverted index hash table can also be used to mining association rules in corpus. In fact, two kinds of hash tables are designed in HI-Apriori: one for generation of candidate frequent itemsets, one for filter of candidate frequent itemsets. Experiments in [13] has shown that while FP-Tree is suitable for mining traditional transaction dataset, inverted index hash table is more efficient for mining corpus.

For distributed computing platforms, the inverted index hash table can be constructed parallelly. Each process will build an inverted index hash table for the allocated portion of the corpus, and for each term, its indexed list might appear in multiple processes, hence, these local inverted index hash tables need to merge into a global inverted index hash table.

C. Generation of frequent itemsets

Like the Apriori algorithms, the HI-Apriori generates frequent k -itemsets from frequent $(k-1)$ -itemsets by the join manipulation. Since the number of combinations grows by square, the join manipulation consumes most of the computation time and is the hot spot, thus, parallelization is utilized to address this issue.

Suppose the frequent $(k-1)$ -itemsets are allocated to a group of processes, then, besides making join with itself, the itemsets within each process need to join with that of other processes. To do so, processes need to exchange itemsets among them using some communication scheme.

III. DESIGN OF PHI-APRIORI

A. Partition of the corpus

Assume that there are n nodes and p processes. Denote by D the corpus, by $|D|$ the number of documents contained in D , and by M the total size of D . Suppose process i is assigned a subset S_i of D that contains r_i documents, $i = 1, \dots, p$, then,

$$\sum_{i=1}^p r_i = |D|, \quad r_i \geq 1 \quad (1)$$

S_i is determined by

$$S_i = \{d_j \in D \mid \sum_{k=0}^{i-1} r_k + 1 \leq j \leq \sum_{k=0}^i r_k, \quad r_0 = 0\} \quad (2)$$

such that

$$|m(S_i) - M / p| \leq \varepsilon, \quad i = 1, \dots, p \quad (3)$$

where $m(S_i)$ is the size of S_i and $\varepsilon > 0$ is a sufficiently small number. However, Eq. (1)-(3) is a NP-Complete problem which requires heavy computational overhead. Thus, the MULTIFIT algorithm [14] for the dual packing problem is adopted here to find a sub-optimal solution. The tasks are then allocated to the processes via MPI_Scatter.

B. Distributed inverted index hash table

After the assignment of workload, each process constructs a local inverted index hash table, as illustrated by the first picture in Figure1. These index hash tables should be merged to a global inverted index hash table for later use. To achieve this, PHI-Apriori collects the bucket split information of local inverted index hash table of each process via MPI_Gather, determines the maximum number of split, and broadcast it to all processes via MPI_Bcast. Each process then splits completely its local inverted index hash table according to this number so that all the local inverted index hash tables have the same entries and buckets, and there is only one entry pointing to each bucket. Notice that, a complete split might results in some empty buckets which cost few spaces and have little impact on performance.

Since local inverted index hash tables are split completely, it is straightforward to merge: just merge the buckets under the same entry. There are two ways to do so. One is that the main process collects all the inverted index hash tables and then merges in the main process. The alternative is that each process will be responsible for the merge of some buckets, sending out buckets of other processes' duty and receiving buckets of its own duty via MPI_Alltotal. Then, each process merges buckets. Finally, each process sends out its merged buckets and receives others merged buckets via MPI_Alltotal, and finally, all processes obtain a copy of the global inverted index hash table.

C. Generation of candidate frequent itemsets

Generation of frequent itemsets begins with frequent 1-itemsets. In the previous step, when a process i has merged the allocated buckets, it is able to generates a frequent 1-itemset $F_i(1)$, whose union $F(1) = \bigcup_{i=1}^p F_i(1)$ composes

the frequent 1-itemset for the corpus. Construction of higher order itemsets needs the cooperation of all processes. To simplify the communications, processes are logically linked as a ring so that each process need only communicates with its neighbors. Assume process i hosts a frequent $(k-1)$ -itemset $F_i(k-1)$. First, $F_i(k-1)$ is joined with itself, after filtering those that do not meet the support requirement, denote the remianing frequent itemsets by $F(F_i(k-1) \times F_i(k-1))$ and put it into the frequent k -itemset $F_i(k)$. Then, communications begin so that frequent $(k-1)$ -itemsets in different process get the chance to join. In the first round, process i sends frequent itemset $F_i(k-1)$ to the next process $(i+1)\%p$ and receives frequent itemset $F_{(i+p-1)\%p}(k-1)$ from process $(i+p-1)\%p$. $F_{(i+p-1)\%p}(k-1)$ is then joined with $F_i(k-1)$, and the generated frequent itemsets are again put into $F_i(k)$. In the subsequent round of communication, instead of the frequent $(k-1)$ -itemset $F_i(k-1)$, process i sends the frequent itemset that it receives in the previous round to its next process, followed by the joining and filtering manipulations. Such communications need $p/2$ rounds. If p is odd, the frequent k -itemsets generated in process i is

$$F_i(k) = \{F(F_i(k-1) \times F_{(i+p-j)\%p}(k-1)) \mid 1 \leq j \leq p/2\} \quad (4)$$

and $F(k) = \bigcup_{i=1}^p F_i(k)$. The situation is a little more complicated when p is even. The first $p/2-1$ rounds of communication is the same as p is odd, in the last round of communication, there is a difference whether p can be divided by 4 or not. If p can be divided by 4, then, in the last round of communication, processes in $\{i \mid 0 \leq i \leq p/2-1\}$ send data, while processes in $\{i \mid 1 \leq i \leq p/2\}$ receive data. If p can not be divided by 4, then, in the last round of communication, only processes with even number send data, while processes with odd number receive data.

Repeating the above procedure, frequent k -itemsets are generated from frequent $(k-1)$ -itemsets, until no new frequent itemset is generated or k reaches some prescribed number.

D. Filtration of candidate frequent itemsets

When two frequent frequent $(k-1)$ -itemsets are joined, a candidate frequent k -itemset is generated and the filter procedure of HI-Apriori is applied to determines if it satisfies

the minimum support requirement with the aid of inverted index hash table. Since each process keeps a copy of the global inverted index hash table, these can be done locally within each process. Note that the filter procedure does not need to store candidate frequent itemsets, tremendous space to store the candidate frequent itemsets are saved.

IV. EXPERIMENTS

The corpus for the experiments are taken from the Sougou Lab [15]. Four datasets, labelled ds-1, ds-2, ds-3 and ds-4, are drawn from the corpus by random sampling with sample size 1,000, 10,000, 50,000 and 100,1000, respectively. Each of the datasets contains, after preprocessing, 15,710, 35,617, 108,496 and 135,527 distinct terms respectively. Two experiments are designed to find out how the distributed inverted index hash table and the parallel generation of frequent itemsets could enhance the performance of the PHI-Apriori algorithm. Each experiment is carried out on the 4 datasets. By varying the numbers of nodes and processes, both the vertical and the horizontal scalability of PHI-Apriori are observed.

A. Experiment platform

The experiments are carried out on the Tianhe II supercomputer in National Supercomputer Center in Guangzhou. The computing nodes are configured as follows.

TABLE I. CONFIGURATION OF THE EXPERIMENT PLATFORM

| Component | Configuration |
|-----------------|--|
| Processor | 2 × Intel Xeon E5-2692 v2 @ 2.20GHz(64bit) 12 cores |
| Cache | L1 cache of 32KB, L2 cache of 256KB, L3 cache of 30720KB |
| Memory | 64GB |
| Address space | 46-bits physical address, 48-bits virtual address |
| interconnection | THExpress-2, the 160Gbps dedicated high-speed network |

According to the previous analysis, the PHI-Apriori algorithm runs a little more efficiently when using odd number processes than even number, moreover, running too many process in a node results in drop down of performance. Thus, each nodes runs 7 processes and each experiment is deployed to odd number of nodes.

B. Experiments on the efficiency of building distributed inverted index hash table

The first experiment is tested in 4 clusters with 1, 3, 5 and 7 nodes respectively. The time cost of constructing the global inverted index hash table distributedly for each datasets and on each clusters are summarized in Fig ..

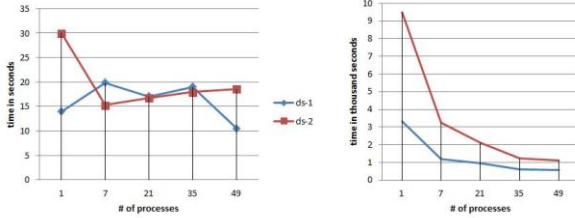


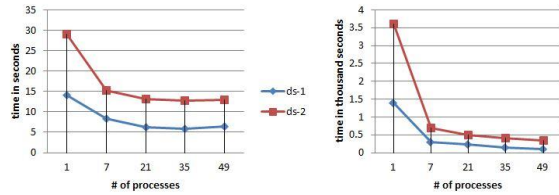
Figure 1. Performance of build distributed inverted index hash table.

Since the time cost for dataset ds-3 and ds-4 is by far greater than that for ds-1 and ds-2, hence, two line charts are drawn instead. In fact, this result indicates that when the dataset is relatively small, constructing a distributed inverted index hash table does not bring much advantages. The reason is that, the time saved by reading and processing dataset by multiple processes is counterbalanced by the time spent on synchronization of the distributed inverted index hash table.

However, when the dataset grows to the scale of 10^5 , the overhead of processing corpus dominates that of synchronization, and the advantages brought by multiple processes begin to appear.

C. Experiments on the efficiency of parallel generation of frequent itemsets

This experiment is carried out for the same dataset and with the same configurations as the first one. Experimental results are collected in Fig. 2.



Performance of parallel generation of frequent itemsets.

TABLE II. BELOW SHOWS SOME MORE DETAILED RESULT FROM THE EXPERIMENTS FOR DS-4.

THE SPEEDUPS OF PHI-APRIORI ON DS-4

| # of nodes | # of processes | Time(s) | Speedup | Memory usage per process(KB) |
|------------|----------------|---------|---------|------------------------------|
| 1 | 1 | 3469 | 1 | 1165422 |
| 1 | 7 | 528 | 6.57 | 853410 |
| 3 | 21 | 348 | 9.97 | 855496 |
| 5 | 35 | 290 | 11.96 | 893648 |
| 7 | 49 | 211 | 16.44 | 887896 |

Obviously, the speedup increases as the number of processes increases. Furthermore, compared to corpus of small size, the speedup gained for corpus of large size using the same number of nodes and processes is bigger. The PHI-Apriori algorithm has good vertical scalability in the sense that, as the size of corpus grows, the speedup increases.

V. CONCLUSION

Mining association rules in corpus is different from transaction database. Based on previous work on the HI-Apriori algorithm, this paper focus on extending HI-Apriori to the high performance computing platforms. Parallelized version for constructing the global inverted index hash table and for generating frequent itemsets are presented. The PHI-Apriori algorithm is then implemented via the standard MPI library. Experimental results on several combination of different corpus and different clusters demonstrate that PHI-Apriori achieves satisfactory horizontal scalability as well as vertical scalability.

ACKNOWLEDGMENT

This work is supported by National Supercomputer Center in Guangzhou (No. 2013Y2-00036).

REFERENCES

- [1] The 33rd Statistic Report on Internet Development in China. <http://www.cnnic.net.cn/hlwfzyj/hlwzxbg/hlwjtjbg/201403/t20140305-46240.htm>.
- [2] Mills M. T., Bourbakis N. G., Graph-Based Methods for Natural Language Processing and Understanding—A Survey and Analysis, IEEE Transactions on Systems, Man and Cybernetics Part C: Applications and Reviews :59–71, 2013.
- [3] Muntean M., Cabulea L., Vslean H., A new text clustering method based on Huffman encoding algorithm, Proceedings of 2014 IEEE International Conference on Automation, Quality and Testing, Robotics, Cluj-Napoca, pp. 1–6, 2014.
- [4] Han J., Wang C., El-Kishky A., Bringing structure to text: Mining phrases, entities, topics, and hierarchies, Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, New York, pp. 1968–1968, 2014.
- [5] Poelmans J., Ignatov D. I., Kuznetsov S. O., Dedene G., Formal concept analysis in knowledge processing: A survey on applications, Expert Systems with Applications :6538–6560, 2013.
- [6] Poelmans J., Ignatov D. I., Kuznetsov S. O., Dedene G., Formal concept analysis in knowledge processing: A survey on models and techniques, Expert Systems with Applications :6601–6623, 2013.
- [7] El Oirrak A., Aboutajdine D., Combining BOW representation and Apriori algorithm for text mining, 5th International Symposium on I/V Communications and Mobile Networks, Rabat, pp. 1–4, 2010.
- [8] Reddy G. S., Rajinikanth T. V., Rao A.A., A frequent term based text clustering approach using novel similarity measure, Souvenir of the 2014 IEEE International Advance Computing Conference, Gurgaon, pp. 495–499, 2014.
- [9] Zhang Z., Zhu H., TOP-N most frequent item set mining algorithm based on improved inverted list and set theory, Journal of Computational Information Systems., :6261–6271, 2014.
- [10] Mooney R. J., Roy L., Content-based book recommending using learning for text categorization, Proceedings of the Fifth ACM Conference on Digital Libraries, San Antonio, pp. 195–204, 2000.
- [11] Babashzadeh A., Daoud M., Huang, J., Using semantic-based association rule mining for improving clinical text retrieval, Lecture Notes in Computer Science., :186–197, 2013.
- [12] Agrawal R., Shafer J. C., Parallel Mining of Association Rules, IEEE Transactions on Knowledge and Data Eng., :962–969, 1996.
- [13] he Sougou Corpus. <http://www.sogou.com/labs/>.