This is the published version:


Available from Deakin Research Online:

[http://hdl.handle.net/10536/DRO/DU:30072371](http://hdl.handle.net/10536/DRO/DU:30072371)

Every reasonable effort has been made to ensure that permission has been obtained for items included in Deakin Research Online. If you believe that your rights have been infringed by this repository, please contact [drosupport@deakin.edu.au](mailto:drosupport@deakin.edu.au)

**Copyright**: 2014, ICIC International
A NOVEL PARALLEL ALGORITHM FOR FREQUENT ITEMSETS MINING IN MASSIVE SMALL FILES DATASETS

DAWEN XIA1,2, ZHUOBO RONG1, YANHUI ZHOU1, YANTAO LI1, YU SHEN1 AND ZILI ZHANG1,3,*
1School of Computer and Information Science
Southwest University
No. 2, Tiansheng Road, Beibei District, Chongqing 400715, P. R. China
gzmdxdw@swu.edu.cn

2School of Information Engineering
Guizhou Minzu University
Huaxi District, Guiyang 550025, P. R. China

3School of Information Technology
Deakin University
VIC 3217, Australia
*Corresponding author: zhangzl@swu.edu.cn

Received April 2013; accepted July 2013

ABSTRACT. In big data analysis, frequent itemsets mining plays a key role in mining associations, correlations and causality. Since some traditional frequent itemsets mining algorithms are unable to handle massive small files datasets effectively, such as high memory cost, high I/O overhead, and low computing performance, we propose a novel parallel frequent itemsets mining algorithm based on the FP-Growth algorithm and discuss its applications in this paper. First, we introduce a small files processing strategy for massive small files datasets to compensate defects of low read-write speed and low processing efficiency in Hadoop. Moreover, we use MapReduce to redesign the FP-Growth algorithm for implementing parallel computing, thereby improving the overall performance of frequent itemsets mining. Finally, we apply the proposed algorithm to the association analysis of the data from the national college entrance examination and admission of China. The experimental results show that the proposed algorithm is feasible and valid for a good speedup and a higher mining efficiency, and can meet the actual requirements of frequent itemsets mining for massive small files datasets.

Keywords: Big data analysis, Frequent itemsets mining, Parallel FP-Growth, Small files problem, Hadoop MapReduce

1. Introduction. In the context of big data [1,2], as an important means of data statistics and analysis, data mining can help us to find the association relationships among the exponential growth of massive data reasonably and efficiently; therefore, it is becoming an urgent need for high profit enterprises in the business intelligence analysis.

As one of the important research directions in data mining, frequent itemsets mining plays an essential role in mining associations [3], correlations [4], causality [5] and other important data mining tasks [6], which is a strong impetus to the applications of association rules in market selection, decision analysis and business management [7-9]. The existing classical frequent itemsets mining algorithms are the breadth-first algorithm Apriori proposed by Agrawal and Srikant in 1994 [3] and the depth-priority algorithm FP-Growth presented by Han et al. in 2000 [6]. In order to improve the efficiency of frequent itemsets mining, many researchers have proposed several methods to optimize the classical algorithms [10-15]. Meanwhile, to break the bottleneck of mining performance and reduce memory consumption and computational cost of the machine in the single machine
environment, some parallel and distributed algorithms are proposed [16-22]. However, for
traditional methods, the applications of corresponding algorithms for frequent itemsets
mining in the large-scale datasets will easily cause high CPU consumption, high memory
cost, high I/O overhead, low computing performance and other issues.

To meet the rapidly growing demands of large-scale data processing, Apache Software
Foundation has launched Hadoop framework based on the open-source implementation of
the Google File System (GFS [23]) and programming model (MapReduce [24]) in 2006,
including Hadoop Distributed File System (HDFS [25]) and Hadoop MapReduce, etc. As
a typical method, the Hadoop framework provides a new idea for handling big data. In
the frequent itemsets mining for large-scale data, a parallel FP-Growth (PFP) algorithm
is proposed by MapReduce approach in [26], and the performance of PFP algorithm is
enhanced by adding load balance feature in [27]. All the previous methods have many
desirable properties, but these methods ignore frequent itemsets mining for massive small
files datasets in Hadoop.

With the arrival of the big data era, massive data are growing rapidly. However, in
reality, most of the large-scale data are composed of massive small files (e.g., financial and
investment data, business transaction data, and Web log data). Small files are usually
referred to as the files with sizes less than 64 MB that is the default size of the HDFS
data block. According to a study in 2007 at the National Energy Research Scientific
Computing Center, 43% of the over 13 million files in a shared parallel file system are under
64 KB and 99% are under 64 MB [28], and more scientific applications that consist of a
large number of small files are depicted in [29]. Nevertheless, in the face of massive small
files datasets, the constructed FP-tree in parallel FP-Growth (PFP) algorithm cannot fit
into the memory, which often causes problems such as memory overflow and huge com-
unication overhead. Furthermore, Hadoop was originally designed to process streaming
large files, which has inherent defects in handling massive small files that will decrease
the access efficiency of HDFS and increase the additional overhead of MapReduce, while
the computing efficiency of the Hadoop platform largely depends on the performance of
HDFS and MapReduce [30]. However, the massive small files processing will reduce the
overall performance of the entire mining task, which is mainly shown in the following two
aspects. (1) The access efficiency of HDFS is decreased, as the Namenode is responsible
for managing and scheduling huge amounts of metadata stored in Datanodes, which needs
to search and retrieve the requested file blocks among Datanodes with a large number of
search operations. Thus, a lot of memory space of Namenode is occupied, which will be
bound to lead to inefficient data access. (2) The additional overhead of MapReduce is
increased, as a Map task usually executes one input data block once only. If the files are
too small in size and numerous in number, lots of Map tasks will be generated and each
Map task deals with a small amount of data, which will inevitably increase the additional
overhead of MapReduce.

In this paper, we propose a novel parallel algorithm for frequent itemsets mining based
on the FP-Growth algorithm and discuss its applications. In particular, we introduce a
small files processing strategy in the FP-Growth algorithm, to compensate defects of
low read-write speed and low processing efficiency for handling the massive small files
datasets in Hadoop, and to enhance the access efficiency of HDFS and reduce the addi-
tional overhead of MapReduce. On the other hand, we use MapReduce to implement the
parallelization of FP-Growth algorithm, thereby improving the overall performance and
efficiency of frequent itemsets mining.

The remainder of this paper is organized as follows. First, the basic theories are briefly
reviewed in Section 2. Then, the proposed algorithm is described in detail in Section
3, and the implementation of the proposed algorithm is depicted in Section 4. Next,
the results of several experiments and their analysis are presented in Section 5, and the
actual applications of the proposed algorithm are given in Section 6. Finally, the paper is concluded in Section 7.

2. Preliminaries. In this section, we briefly review the related concepts and definitions of association rules, and FP-Growth algorithm.

2.1. Association rules. Association rule mining is used to reveal the unknown interdependence among the data, and discover lots of interesting association or correlation relationships among a large set of data items [31]. It can be decomposed into two subproblems. (1) Find all frequent itemsets that have support greater than the user-specified minimum support threshold (called min_sup) from the transaction database. (2) Generate the desired rules from the frequent itemsets found by (1); if confidence satisfies the user-specified minimum confidence threshold (called min_conf), the strong association rules are generated. The overall performance of mining association rules is determined by the first step, so the key of the aforementioned problems is to find all frequent itemsets.

**Definition 2.1.** Let \( D = \{T_1, T_2, \ldots, T_n\} \) be a database which contains a set of transactions. That is, \( T_i (i \in [1, \ldots, n]) \) is a transaction which contains a set of items in \( I \). Let \( I = \{I_1, I_2, \ldots, I_m\} \) be a set of items, where each transaction \( T \) is a set of items such that \( T \subseteq I \). That is, \( T_i = \{I_{k1}, I_{k2}, \ldots, I_{kj}\} \) and \( I_{kj} \in I, 1 \leq j \leq k(i), 1 \leq i \leq n \). Let \( A \) be a set of items, a transaction \( T \) contains \( A \) if and only if \( A \subseteq T \). An association rule is an implication of the form \( A \Rightarrow B \), where \( A \subseteq I, B \subseteq I \), and \( A \cap B = \emptyset \).

**Definition 2.2.** Support and confidence are two measures of rule interestingness that respectively reflect the usefulness and certainty of discovered rules. The rule \( A \Rightarrow B \) has support \( s \) and confidence \( c \) in the transaction set \( D \), where \( s \) is the percentage of transactions in \( D \) that contain \( A \cup B \); \( c \) is the percentage of transactions in \( D \) that contain \( A \) also containing \( B \), that is the conditional probability \( P(B|A) \). The support of \( A \) is the percentage of transactions in \( D \) containing the itemset \( A \), that is, \( \text{support}(A) = \{T_i | 1 \leq i \leq n \land A \subseteq T_i \land T_i \in D\} \). More formally, support and confidence are defined as:

\[
\text{support}(A) = \frac{\text{supportcount}(A)}{\text{count}(D)}
\]

\[
\text{support}(A \Rightarrow B) - P(A \cup B) = \frac{\text{supportcount}(A \cup B)}{\text{count}(D)} - \frac{\text{supportcount}(A)}{\text{supportcount}(A)}
\]

\[
\text{confidence}(A \Rightarrow B) = \frac{P(B|A) - \text{support}(A \Rightarrow B)}{\text{support}(A)} = \frac{\text{supportcount}(A \cup B)}{\text{supportcount}(A \cup B) - \text{supportcount}(A)}
\]

**Definition 2.3.** When the support and confidence are no less than the min_sup and min_conf respectively, that is support \( (A \Rightarrow B) \geq \text{min_sup} \) and confident \( (A \Rightarrow B) \geq \text{min_conf} \), the association rules are considered to be effective.

2.2. FP-growth algorithm. Apriori is an original Boolean association rules algorithm for mining frequent itemsets, but it has the defects of repeated database scans and high I/O overhead. While the FP-Growth algorithm uses divide-and-conquer method instead of generation-and-test approach [6], it has higher efficiency than Apriori algorithm. First, a large database is compressed into a frequent pattern tree (FP-tree) which still contains the complete information of database in relevance to frequent pattern mining. Subsequently, the FP-tree is divided into a group of conditional pattern bases. Finally, each conditional pattern database is mined respectively.

FP-Growth applies a pattern growth method, which avoids costly candidate generation and test by successively concatenating frequent 1-itemset found in the FP-trees, and it only needs exactly two scans of the transaction database. The first generates frequent 1-itemsets. The second deletes the infrequent items of each transaction in database by using frequent 1-itemsets in the previous step, then constructs the FP-tree, and generates frequent itemsets with FP-tree.
3. Algorithm Description. In this section, we propose a novel parallel algorithm for mining frequent itemsets in massive small files datasets in detail. The flowchart of the proposed algorithm is shown in Figure 1.

(1) Write a small files processing program — *Sequence File*. The *Sequence File* is used to merge all massive small files, which are composed of a large number of transaction datasets stored in HDFS, into a large transaction data file (transaction database).

(2) Equally divide the transaction database into several sub-transaction databases, and then assign them to different nodes in the Hadoop cluster. This step is automatically operated by HDFS, and when necessary we can use the balance command to enable its file system to achieve load balancing.

(3) Compute *support* count of each item in the transaction database by MapReduce, and then obtain the set of *List* from *support* count in a descending order.

(4) Divide *List* into *M* groups, denoted as *Group_list* (abbreviated as *G_list*), and then assign *group_id* for each group sequentially and each *G_list* contains a set of items.

(5) Complete the parallel computing of FP-Growth algorithm by MapReduce. ① The *Map* function (mapper tasks) compares the item of each transaction in the sub-transaction database with the item in *G_list*. If they are the same, then the corresponding transaction is distributed to the machine associated with *G_list*. Otherwise, the *Map* function compares with the next item in *G_list*. Eventually, the independent sub-transaction databases corresponded to *G_list* will be produced. ② The *Reduce* function (reducer tasks) recursively computes the independent sub-transaction databases generated in step ①, and then constructs the FP-tree. This step is similar to the process of traditional FP-tree generation, but the difference is a size *K* max-heap *HP* that stores frequent pattern of each item.

(6) Aggregate the local frequent itemsets generated from each node in the cluster by MapReduce, and finally get the global frequent itemsets.

4. Algorithm Implementation. In this section, we implement the proposed algorithm, which is mainly composed of four steps as described in the following. Note that all pseudo codes are omitted here.
Step 1: Merge massive small files. Sequence File consists of a series of (key, value), where the key is the name of small files and the value is the content of small files before merging. Sequence File uses three classes, i.e., WholeFileInputFormat, WholeFileRecordReader and SmallFilesToSequenceFileConverter, to merge the massive small files into a large file. (1) WholeFileInputFormat Class: The isSplitable() method overloads and returns the value, "false", to maintain the input file not to be divided into slices. The getRecordReader() method returns a customized RecordReader. (2) WholeFileRecordReader Class: the FileSplit is converted into a record, where the key of the record is the filename and the value is the content of this file. (3) SmallFilesToSequenceFileConverter Class: Massive small files are merged into a sequence file, and this class contains the Map() and the Reduce(). The input format of data is WholeFileInputFormat, while the output format is SequenceFileOutputFormat.

Step 2: Compute Llist. The complexity of time and space is O (TDBsize/P), (TDB-size: the size of transaction database, P: the number of parallel programs).

Step 3: Generate G.list from Llist, and complete parallel computing of FP-Growth algorithm. Map() judges that G.list the item of transactions in the machine belongs to, and then sends this transaction to the corresponding G.list machine. In order to avoid sending duplicate transaction, we delete the duplicate entries in a hash table. Each Reduce() handles the independent sub-transaction database associated with G.list, which creates heap HP with a size of K for each item in G.list.

Step 4: Aggregate local frequent itemsets generated in the previous step from each node, and then we get global frequent itemsets.

5. Experiments and Main Results. In this section, we evaluate the proposed algorithm by several experiments, and compare it with PFP algorithm. The main results and their analysis are shown in Section 5.2.

5.1. Experimental setup. The experimental platform based on the Hadoop cluster is composed of one Master machine and four Slave machines, with Intel Pentium (R) Dual-Core E5700 3.00GHz CPU and 2.00GB RAM. All the experiments are performed on Ubuntu 12.04 OS with Hadoop 0.20.2 and Jdk 1.6.0. At the same time, the real data from the Frequent Itemset Mining Dataset Repository are used as the experimental data sources, which are processed into three groups of different datasets, and each group dataset consists of a large number of small files. The composition of the detailed datasets is shown in Table 1.

<table>
<thead>
<tr>
<th>Datasets</th>
<th>Number (each file with less than 64KB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Datasets1</td>
<td>2115 small files</td>
</tr>
<tr>
<td>Datasets2</td>
<td>4218 small files</td>
</tr>
<tr>
<td>Datasets3</td>
<td>8583 small files</td>
</tr>
</tbody>
</table>

5.2. Experimental evaluation. Since the feasibility and validity are the basic requirements of a good algorithm, and the speedup and efficiency are important indicators to measure the stability and merits of an algorithm, we evaluate the overall performance of the proposed algorithm on feasibility, validity, speedup and efficiency.

(1) Feasibility and Speedup Evaluation of the Proposed Algorithm. We evaluate the feasibility and speedup of the proposed algorithm in real datasets of different sizes (Datasets1, Datasets2 and Datasets3) with the number of Datanodes increasing in the cluster. The experimental results are shown in Figure 2.

1http://fimi.ua.ac.be/data/
As we can see from Figure 2, with massive small files datasets, the proposed algorithm can normally complete distributed computing and accurately find the frequent itemsets in the MapReduce environment, which shows that the proposed algorithm is feasible. Especially, with the number of Datanodes increasing gradually, the running time of the proposed algorithm is decreasing proportionally while the overall performance is improving multiply, which indicates that the proposed algorithm has a good speedup.

(2) Validity and Efficiency Evaluation of the Proposed Algorithm. We analyze comparatively the validity and efficiency of the proposed algorithm and the PFP algorithm, which are used respectively for frequent itemsets mining in three datasets of different sizes (Datasets1, Datasets2 and Datasets3). The experimental results are shown in Figure 3.

![Figure 2. The feasibility and speedup of the proposed algorithm](image1)

![Figure 3. The validity and efficiency of the proposed algorithm](image2)

As we can see from Figure 3, when the cluster in the pseudo-distributed environment (only one Datanode) switches to a fully distributed environment (more than two Datanodes), the processing performance of the proposed algorithm is enhanced significantly, which shows that the proposed algorithm is valid. Most importantly, when the number of Datanodes is increasing gradually, the running time of the proposed algorithm is always less than that of PFP algorithm, which indicates that the proposed algorithm has a higher mining efficiency than PFP algorithm.

The proposed algorithm is migrated to the MapReduce environment, which implements the parallel computing of FP-Growth algorithm, therefore, enhances the overall performance of frequent itemsets mining. More specifically, introducing the small files processing strategy, the proposed algorithm compensates inherent defects of Hadoop in handling massive small files datasets, saves memory cost greatly, improves the efficiency of data access, avoids memory overflow, and reduces I/O overhead, thereby improving the mining efficiency of PFP algorithm.

6. Applications. In this section, the actual applications of the proposed algorithm are presented. The frequent itemsets mining for massive small files datasets has wide practical applications, and the proposed algorithm can be applied to the association analysis of the data from the national college entrance examination and admission of China, in addition to mining frequent itemsets in financial and investment data, business transaction data, Web log data and so on.

In this paper, the proposed algorithm is used to analyze the unequal strength association of colleges, majors, areas from the real massive small files datasets of the national college entrance examination and admission of China, which are composed of candidates basic data, recruiting application data, major setting data and admission result data from the
year 2003 to 2012 in a province. We have drawn some important and valuable results as follows. (1) There is a strong association relationship among Southwest University, Chongqing University of Posts and Telecommunications, and Southwest University of Political Science and Law; and it is the same with Guizhou Minzu University and Guizhou University of Finance and Economics. (2) Colleges that lay in the coastal cities have greater popularity, and the admission scores of colleges in the North China are much higher than that in the Northwest. (3) The Administration Management major of a college is the most popular, and English and International Economy and Trade major are quite popular as well.

When filling out college recruiting applications, candidates should maintain a certain gradient under the same circumstance, and try to avoid applying for the colleges and majors with strong association and the colleges in those areas with greater popularity, so that they will obtain more admission opportunities. At the same time, in the association analysis of the data from the national college entrance examination and admission of China, the proposed algorithm shows better processing performance and a higher mining efficiency than PFP algorithm.

7. Conclusions and Future Work. In this paper, aiming at solving the existing problems of traditional frequent itemsets mining algorithms for handling massive small files datasets, we propose a novel parallel algorithm based on the FP-Growth algorithm, and implement its applications. Furthermore, we use three groups of real massive small files datasets to evaluate the feasibility, speedup, validity and efficiency of the proposed algorithm, and analyze PFP algorithm comparatively. The experimental results show that the proposed algorithm can make a breakthrough where PFP algorithm has its defects in processing massive small files datasets, and can meet the actual needs of frequent itemsets mining for massive small files datasets, with a good speedup and a higher mining efficiency. In the future, we will do further research on grouping algorithm and consider the applications of grouping balance strategy and loading balance strategy in the proposed algorithm.

Acknowledgment. The authors would like to thank the anonymous reviewers for their helpful comments and suggestions. This work is partially supported by the National Science and Technology Support Program of China (No. 2012BAD35B08), the Science and Technology Foundation of Guizhou (No. LKM201212), the Fundamental Research Funds for the Central Universities (No. XDJK2013C116), and the Scientific and Technological Innovation Fund of School of Computer and Information Science of Southwest University (No. SWUJX2013-6).

REFERENCES


