

# A non-group parallel frequent pattern mining algorithm based on conditional patterns\*

Zhe-jun KUANG<sup>†1</sup>, Hang ZHOU<sup>†2</sup>, Dong-dai ZHOU<sup>†‡3</sup>, Jin-peng ZHOU<sup>4</sup>, Kun YANG<sup>5</sup>

<sup>1</sup>College of Computer Science and Technology, Changchun University, Changchun 130022, China

<sup>2</sup>School of Economics, Changchun University, Changchun 130022, China

<sup>3</sup>School of Information Science and Technology, Northeast Normal University, Changchun 130117, China

<sup>4</sup>Division of Engineering Science, University of Toronto, Ontario M5S2E8, Canada

<sup>5</sup>School of Computer Science and Electronic Engineering, University of Essex, Colchester CO43SQ, UK

<sup>†</sup>E-mail: kuangzhejun@ccu.edu.cn; zhouhang0311@163.com; ddzhou@nenu.edu.cn

Received Aug. 5, 2018; Revision accepted Dec. 18, 2018; Crosschecked Aug. 23, 2019

**Abstract:** Frequent itemset mining serves as the main method of association rule mining. With the limitations in computing space and performance, the association of frequent items in large data mining requires both extensive time and effort, particularly when the datasets become increasingly larger. In the process of associated data mining in a big data environment, the MapReduce programming model is typically used to perform task partitioning and parallel processing, which could improve the execution efficiency of the algorithm. However, to ensure that the associated rule is not destroyed during task partitioning and parallel processing, the inner-relationship data must be stored in the computer space. Because inner-relationship data are redundant, storage of these data will significantly increase the space usage in comparison with the original dataset. In this study, we find that the formation of the frequent pattern (FP) mining algorithm depends mainly on the conditional pattern bases. Based on the parallel frequent pattern (PFP) algorithm theory, the grouping model divides frequent items into several groups according to their frequencies. We propose a non-group PFP (NG-PFP) mining algorithm that cancels the grouping model and reduces the data redundancy between sub-tasks. Moreover, we present the NG-PFP algorithm for task partition and parallel processing, and its performance in the Hadoop cluster environment is analyzed and discussed. Experimental results indicate that the non-group model shows obvious improvement in terms of computational efficiency and the space utilization rate.

**Key words:** Frequent pattern mining; Parallel algorithm; Conditional pattern bases; MapReduce; Big data

<https://doi.org/10.1631/FITEE.1800467>


**CLC number:** TP301

## 1 Introduction

The rapid development of information technology has ushered in the era of big data, driven by the spreading use of the Internet, social networks, security, and the Internet of Things (Wang et al., 2015; Zhang et al., 2016; Zhuang et al., 2017). Data in the network space are being generated and stored at unprecedented and increasing rates. The emergence of big data has important implications for many social agents, including industry, academia, and

<sup>‡</sup> Corresponding author

\* Project supported by the Fundamental Research Funds for the Central Universities, China (No. 2412015KJ005), the Twelfth Five-Year Plan Project of the Education Department of Jilin Province, China (No. 557), and the Thirteenth Five-Year Plan for Scientific Research of the Education Department of Jilin Province, China (No. JJKH20191197KJ)

 ORCID: Zhe-jun KUANG, <http://orcid.org/0000-0001-5632-7596>; Dong-dai ZHOU, <http://orcid.org/0000-0001-5053-2935>

© Zhejiang University and Springer-Verlag GmbH Germany, part of Springer Nature 2019

government agencies (Chen et al., 2013; Zhao et al., 2016; Zhang et al., 2017). Big data are generated automatically by humans, machines, and material sources, all of which intersect and become interrelated in cyberspace. At present, data sources are increasing rapidly, data are being renewed and updated more frequently, and the scale of the data available is expanding inexorably. However, owing to the problems of data sparsity, traditional technologies and methods are inadequate for effective and quick data acquisition requested by a specific user. The contradictory situation therefore arises that we have access to a massive amount of data, yet there is a lack of knowledge.

Three major types of traditional algorithms have been derived from the association rule mining algorithms: the Apriori algorithm (Agrawal and Srikant, 1994), the frequent pattern (FP) algorithm (Han et al., 2000), and the Eclat algorithm (Zaki, 2000). However, a large number of candidate itemsets have been generated by the iterative processes in the Apriori and Eclat algorithms. The input/output (I/O) overhead increases exponentially with the increase in computation as a result of multiple scans of datasets. Frequent pattern mining is the core algorithm of association rule mining and determines the frequent itemset from the itemset of the frequency variable. The FP algorithm follows the method of FP-growth, without generating candidate patterns. Only two scans of the database are required by the FP algorithm. The concealed relationship in the dataset can be identified from the frequent itemset. Nevertheless, as the dataset grows, a commensurately large memory space is required to build the large FP-tree.

Traditional algorithms are unable to meet the requirements of big data, because the generation of such data has placed increasing demands on FP mining. Li et al. (2008) proposed the parallel frequent pattern (PFP) mining algorithm, which can perform processing using the MapReduce model in the Hadoop platform. The transaction dataset is reorganized by the PFP algorithm group list, decomposing the data and allowing them to be addressed by parallel processing in cluster computing. However, to ensure that the associated rule is not destroyed during task partitioning and parallel processing, the inner-relationship data must be protected. Because redundant storage of this data is required, the space

usage will increase significantly, in comparison with the original dataset.

In this study, we present the non-group parallel frequent pattern (NG-PFP) mining algorithm for task partition and parallel processing in detail, and analyze and discuss its performance in the Hadoop cluster environment. Three MapReduce processes are required in the NG-PFP algorithm, the key idea being that the local FP-tree is built on the basis of the global frequent item list (F-list). This ensures that the inner-relationship between the frequent items will not be changed when the task shard is processed on different machines. Our algorithm differs from the PFP algorithm in that PFP is resolved in the transaction database of the group list (Section 3.2). If the transaction data contain the same items that exist in different groups, those transaction data will be redelivered to multiple groups in whole or in part. Conversely, in the NG-PFP algorithm, the original transaction database is decomposed and run by the FP-growth algorithm. Hence, we can eliminate the grouping, and there is no longer a need to resolve the transaction dataset in multiple redundant parts.

The NG-PFP algorithm has two additional advantages. First, in contrast with the PFP algorithm, the non-group mechanism can reduce the redundant data during the process of the algorithm. Second, task partitioning is related to the size of the task, and does not strongly depend on the group list.

For the small cluster test environment, we used the WebDocs, a real-life huge transactional dataset (Lucchese et al., 2004), as the test set to evaluate the computing space occupancy rate and computational efficiency. We used nine machines as the test environment to run the association rule mining algorithm. The results showed that the processing redundancy data decreased from 1.43 GB to 0.94 GB.

## 2 Related work

In the era of data explosion, due to the multi-source, massive, heterogeneous, and dynamic characteristics of application data involved in a distributed environment, the traditional serial algorithms (Agrawal and Srikant, 1994; Srikant and Agrawal, 1996; Han et al., 2000; Zaki, 2000, 2001b; Agarwal et al., 2002; Wang and Han, 2004; Siddiqua et al., 2017) are no longer able to meet the needs of

data mining. Hence, dealing with problems of big data with the parallel association algorithm has become a necessary research task. Some traditional parallel algorithms (Zaki, 2001a; Cong et al., 2005; El-Hajj and Zaïane, 2006) provided a good theoretical support for large data processing. However, the traditional parallel algorithms cannot directly deal with big data. In big data, the processing platform is based on key-value databases. Key-value databases are constituted as a simple data model and data are stored corresponding to key-values. Therefore, traditional parallel algorithms cannot meet the challenges on the categories and scales created by big data.

di-Jorio et al. (2009) proposed a paralleled gradual itemset extraction (GRITE) algorithm, which is based on the linear-time closed itemsets mining (LCM) algorithm; they found that the former showed a linear growth of mining time and that it is applicable to complicated datasets. Liu et al. (2015) proposed the parallel mining method, which can be used for a cloud computing platform. On the compute node resource allocation problems, considering execution efficiency and load balancing, Lin and Chung (2015) proposed FLR-Mining, which is capable of determining the appropriate number of computing nodes automatically and achieving better load balancing compared with existing methods. Compared with the traditional serial algorithms, parallel algorithms can solve the problem of limited memory and effectively improve the performance.

Currently, big data processing depends mainly on parallel programming models like MapReduce, which provide a cloud computing platform of big data services for the public. Some research (Yang et al., 2010; Li et al., 2012; Riondato et al., 2012) focuses on improving the Apriori algorithm based on the MapReduce model. Lin et al. (2012) proposed three algorithms to investigate effective implementations of the Apriori algorithm in the MapReduce framework. Other research (Zhang et al., 2013; Zheng and Wang, 2014; Ge et al., 2017) focuses on enhancing the Eclat algorithm based on the MapReduce model.

In the relational schema, the Apriori algorithm is a typical functional dependency (FD), through the data of frequent model expressing the relationship. However, in the real big data environment, problems often arise concerning inconsistent data or semantically related pattern recognition problems.

Caruccio et al. (2016) extended FD definitions with the term relaxed functional dependencies (RFDs). Caruccio et al. (2017) determined a proper distance threshold for a given relaxed FD holding over the entire database. Kruse and Naumann (2018) presented a novel and highly efficient algorithm, Pyro, to discover both approximate FDs and approximate unique column combinations. Berti-Équille et al. (2018) defined a notion of genuineness and proposed algorithms to compute the genuineness score of a discovered FD. This can be used to identify the genuine FDs within the set of all valid dependencies that hold on the data. Huhtala et al. (1999) presented an efficient algorithm to find functional dependencies from large databases, which respects database attribute values and partitions the rowsets. In addition, Mandros et al. (2017), Bauer et al. (2018), and Caruccio et al. (2018) considered the problem of discovering relaxed functional dependencies; for example, the Apriori and Eclat algorithms depend on the number of data items, and need multiple iterations, thus increasing the computational complexity in the MapReduce frame.

Li et al. (2008) implemented the conventional FP algorithm using the MapReduce method, and achieved algorithm parallelism which not only divides a large dataset into many smaller sub-tasks, but also processes multiple sub-tasks concurrently. Yu et al. (2007), Zhou et al. (2010), and Yang et al. (2016) considered the problem of load balance; they presented balanced parallel FP-growth algorithms that are based on the PFP algorithm and improved parallelization, and thereby improved the performance. Xia et al. (2013, 2014) used MapReduce to implement the parallelization of the FP-growth algorithm, thus improving the overall performance of frequent itemset mining. Deng and Lou (2015) presented a distributed SPFP algorithm based on the Spark framework and improved the FP-growth algorithm. However, these parallel FP-like algorithms establish a group list (Q-list) after obtaining a frequent patterns list, and redeliver transaction data followed by a Q-list. Meanwhile, in the processes of task partitioning and parallel processing, the associated rule must not be destroyed and a great amount of redundant data are needed to protect inner-relationships between sub-tasks, thus leading to a larger storage space compared with the original dataset.

Unlike the aforementioned research, we present

the non-group PFP (NG-PFP) mining algorithm in the MapReduce framework, in which we focus on cancelling the grouping mechanism rather than reorganizing transaction data based on a group list.

### 3 Problem statement

Mining out all the frequent patterns is a main challenge in association rule mining. In this study, we are concerned with minimum support, because it could control the scope for the FP set. The FP problem is defined below. Let  $I = \{i_1, i_2, \dots, i_m\}$  denote an itemset and  $D$  a transaction dataset, where each transaction  $t$  is an itemset ( $t \subseteq I$ ). Each transaction has a unique identifier TID. If  $X \subseteq t$ , transaction  $t$  includes a subset  $X$  of  $I$ . The association rule is an implication from  $X \Rightarrow Y$ , where  $X \subset I, Y \subset I$ , and  $X \cap Y = \emptyset$ .  $N$  is the total number of itemsets.  $A$  is an itemset and  $A \subset I$ ; when  $\text{support}(A) = A/N, \text{support}(A) \geq \epsilon$  (the minimum support) and  $A$  is a frequent pattern set.

#### 3.1 Missing pattern problem

In a big data environment, the process of FP mining is conducted using the MapReduce model. In this model, a heavy task is decomposed into many sub-tasks, after which parallel processing is performed to obtain the result that combines all answers for the sub-tasks. The FP algorithm is different from the Apriori algorithm in the MapReduce model of parallel processing. Given a large number of candidate sets and multiple scans at the database conducted via the Apriori algorithm, the association rules and FPs will not miss even the random sub-tasks. However, the FP algorithm obtains the FP sets by scanning the database only twice; the FP and association rules are compressed in the global FP-tree. When random decomposition of heavy tasks is performed using the FP algorithm in the MapReduce model, some FPs may not be identified.

Assume that the transaction database is shown in Table 1 and that the minimum support threshold value is three. We can obtain the frequent item count by first scanning the database. According to the definition of frequent items and assumptions, frequent items should meet the conditions in which the frequent count is no less than three. The frequent-list (F-list) and FP-tree for Table 1 are shown in Fig. 1. By following the FP algorithm, we can obtain the

FP-tree by referring to the F-list and performing a second scan of the database. Then the transaction database is decomposed into three shards, i.e.,  $\langle f, c, a, m, p \rangle$  and  $\langle f, c, a, b, m \rangle$ ,  $\langle f, b \rangle$  and  $\langle c, b, p \rangle$ , and  $\langle f, c, a, m, p \rangle$  (Table 2).

Table 1 Transaction dataset

TID	Original dataset
100	{f, a, c, d, g, i, m, p}
200	{f, a, b, c, i, m, o}
300	{b, f, h, j, o}
400	{b, c, k, s, p}
500	{a, f, c, e, p, m, n}

Table 2 Shard transformation data

Shard ID	Transaction
Shard 1	{f, c, a, m, p} and {f, c, a, b, m}
Shard 2	{f, b} and {c, b, p}
Shard 3	{f, c, a, m, p}

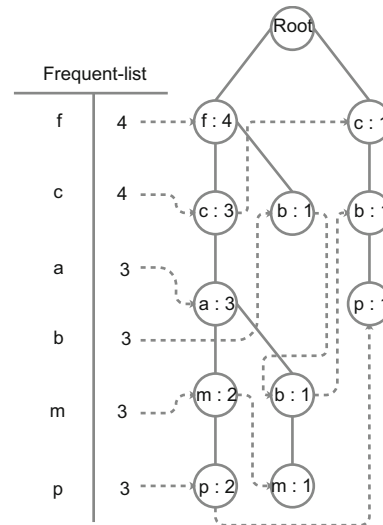


Fig. 1 Frequent-list and FP-tree for Table 1

The local frequency counts for item f are 2, 1, and 1. Although the global frequency count of f is 4, which is greater than the minimum support threshold ( $\epsilon = 3$ ), the local frequency count is less than the minimum support threshold. Given that the frequent items are distributed in the different shards, thus making the local frequency count less than the threshold, the system may be deemed having no frequent items in the local shard. Therefore, the frequency pattern f will probably be missing in sub-task processing.

### 3.2 Parallel frequent pattern algorithm

The PFP algorithm performs paralleled computation with a three-phase MapReduce in five steps as follows:

1. The transaction dataset is separated into shards, which are stored on multiple computers.
2. Start to compute the count for frequent itemsets in the transaction dataset, define the F-list, and then sort by the count.
3. The frequent items in the F-list are grouped into  $Q$  groups, and this new list is called the Q-list. Each group of frequent items has a unique group identifier (gid). The grouping process requires a small amount of calculation, so the process is completed on a single machine.
4. Next, the Q-list is partitioned, and the local FP-tree in each task is formed, followed by the FP mining of the sub-tasks.
5. Combining and de-weighting are performed to obtain the full set of FPs.

The PFP algorithm process is shown in Fig. 2. In this process, step 4 is considered the most critical step. The transaction dataset shown in Fig. 1 is divided into three groups:  $\langle \text{Group1: } f, c \rangle$ ,  $\langle \text{Group2: } a, b \rangle$ , and  $\langle \text{Group3: } m, p \rangle$ . The PFP grouping strategy performs horizontal sharding of the conventional FP-tree and then duplicates and migrates the transaction data according to the group list. Therefore, no frequent items will be omitted during the FP mining process as a result of the local frequent items support being less than the minimum threshold value. Meanwhile, the sharded mining tasks guarantee that the local transaction data are less than the original data, thus facilitating parallel processing and improving mining efficiency. However, some problems may arise in step 4 of the PFP algorithm. In particular, parallel processing on the basis of the group list may achieve breakthrough improvements in time validity, but can also lead to wasted space. For example, in one transaction data  $\langle f, c, a, b, m, p \rangle$  are recognized as three parts  $\langle f, c, a, b, m, p \rangle$ ,  $\langle f, c, a, b \rangle$ , and  $\langle f, c \rangle$ , and then assigned to three different groups:  $\langle \text{Group1: } f, c, a, b, m, p \rangle$ ,  $\langle \text{Group2: } f, c, a, b \rangle$ , and  $\langle \text{Group3: } f, c \rangle$  (Fig. 3). Although step 4 has been cut via the hash table, it does not solve the essential problems related to the wasted space. Assuming that each letter takes a unit space,  $\langle f, c, a, m, p \rangle$  occupies five

unit spaces, replicated and assigned to three different groups as described above. Groups 1, 2, and 3 take up to 2, 3, and 5 unit spaces, respectively, adding up to 10 unit spaces. Hence, after decomposing the data, the space occupancy rate becomes two times larger than the original data. With the increase in the number of groups and length of the transaction data, the space occupancy rate becomes larger than the original data. In other words, we have to reserve a space that is several times larger than the big data for the association rule mining process, which is not reasonable.

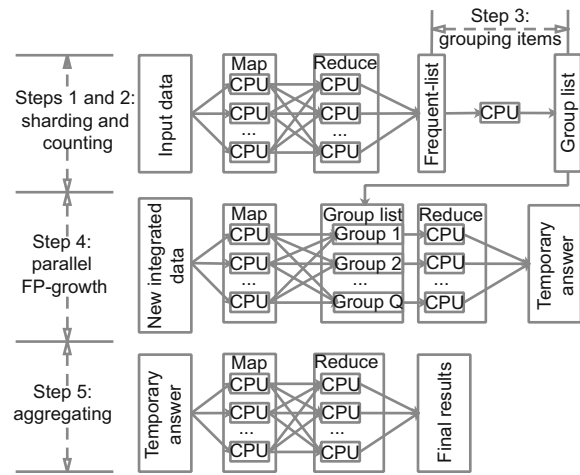


Fig. 2 Parallel frequent pattern algorithm process

Group	Q-list	Transaction data
Group 1	f:4	$\langle f, c \rangle, \langle f, c \rangle, \langle f \rangle, \langle f, c, a, m, p \rangle$
	c:4	$\langle c \rangle, \langle f, c \rangle, \langle f, c, a \rangle, \langle f, c, a, m, p \rangle$
Group 2	a:3	$\langle f, c, a \rangle, \langle f, c, a, b \rangle, \langle f, c, a, m, p \rangle$
	b:3	$\langle f, b \rangle, \langle c, b \rangle, \langle f, c, a, b \rangle, \langle f, c, a, m, p \rangle$
Group 3	m:3	$\langle f, c, a, m, p \rangle, \langle f, c, a, b, m \rangle, \langle f, c, a, m, p \rangle$
	p:3	$\langle c, b, p \rangle, \langle f, c, a, m, p \rangle, \langle f, c, a, m, p \rangle$

Fig. 3 Parallel frequent pattern data transformation

On the other hand, the MapReduce computer system in Hadoop first saves data in the computer and then processes the computation. Hence, there are multiple data migrations in the sharding process, which in turn can reduce computation efficiency. Therefore, as we have considered, we may use a new algorithm based on conditional pattern bases to perform parallel processing without grouping the group



list. This method does not generate redundant data and ensures that frequent items are not omitted in the algorithmic process.

#### 4 Non-group parallel frequent pattern algorithm

The FP algorithm has two main steps: constructing the FP-tree and discovering the conditional pattern base, and then mining the FPs using the FP-growth algorithms. The FP-tree contains the frequent item count and simultaneously maintains the inner-relationship between data. In two instances, the screening is reduced using the minimum support threshold in two main steps. The first is when screening is performed to reduce the infrequent items at the beginning of the created F-list. The second is when screening is performed to reduce the infrequent items in the sequence. Although the PFP algorithm uses the hash function mapping relationship to ensure that the local and global frequency counts are consistent, it will result in a large amount of generated redundant data. Therefore, we propose the NG-PFP algorithm based on a conditional pattern.

Three MapReduce processes are required in the NG-PFP algorithm, and determining the global F-list is the first step in this procedure. The key idea is that the local FP-tree is built on the basis of the global F-list, because we can ensure that the inner-relationship between the frequent items will not be changed after the task shard is processed on different machines. Hence, we can eliminate the grouping, and there is no longer a need to build the local F-list. While screening, we can use the minimum support threshold to reduce the infrequent items in the reducing stage (not in the mapping stage), because the dataset becomes global after the reducing stage. The process of the NG-PFP algorithm is shown in Fig. 4.

1. The global F-list is computed, and transaction data sharding is performed to obtain the frequent computation of all itemsets via MapReduce. The F-list obtained is presented in descending order.

2. The conditional pattern bases of all items are obtained, after which the local FP-tree is formed on the basis of the global F-list. Then the conditional pattern bases of all items are identified using the bottom-up method.

3. The full set of frequent items is obtained, and

the infrequent items in conditional pattern bases are eliminated as per the minimum threshold. Using the FP-growth approach, the full set of frequent items is obtained on the basis of the conditional pattern bases of infrequent items as a minimum threshold.

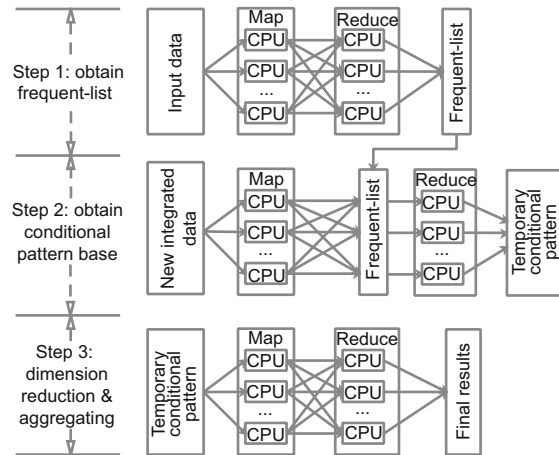


Fig. 4 Process of the non-group parallel frequent pattern (NG-PFP) algorithm

Based on the FP algorithm theory, the main preconditions for FP mining are the F-list and the conditional pattern base. In Section 4.1, we discuss the process of obtaining the F-list in the MapReduce model. The sub-algorithm used to achieve the conditional pattern is discussed in Section 4.2. Finally, the full FP mining process is discussed in Section 4.3.

##### 4.1 F-list count algorithm

Obtaining the F-list is one of the important preconditions for the F-list count algorithm; therefore, the first step is to immediately obtain the F-list. In the MapReduce model, we can improve the Word-Count method for the solution to the F-list. Given that the F-list is constituted mainly by the one-dimensional (1D) itemset and frequency count, each item can be viewed as a single word.

The computation of the frequent item count is conducted using MapReduce, which uses a key-value pair. Here the 1D itemset is regarded as the key, and the frequent count is the value. When the item appears in the transaction data, the value should be “1,” wherein  $a_i$  is the item of the transaction data and  $T_i$  the transaction dataset. Note that  $a_i$  appears in multiple transaction data, forming multiple instances of “1.” Then these instances of “1” are combined to obtain the frequency count of  $a_i$  in all

transaction datasets. The F-list count algorithm is described in Algorithm 1, in which  $S(a_i)$  is the set of the items appearing in all the transaction data and the sum is the frequency count.

**Algorithm 1** F-list count algorithm

**Procedure:**

- 1: Mapper(key, value =  $T_i$ )
- 2: **for** each item  $a_i$  in  $T_i$  **do**
- 3:   Call Output( $\langle a_i, '1' \rangle$ )
- 4: **end for**

**Procedure:**

- 1: Reduce(key =  $a_i$ , value =  $S(a_i)$ )
- 2: Count  $\leftarrow$  0
- 3: min\_support  $\leftarrow$   $\epsilon$
- 4: **for** each item “1” in  $S(a_i)$  **do**
- 5:   Count  $\leftarrow$  Count+1
- 6:   **if** Count  $\geq$  min\_support **then**
- 7:     Call Output( $\langle \text{null}, a_i + \text{Count} \rangle$ )
- 8:   **else**
- 9:     Continue
- 10:   **end if**
- 11: **end for**

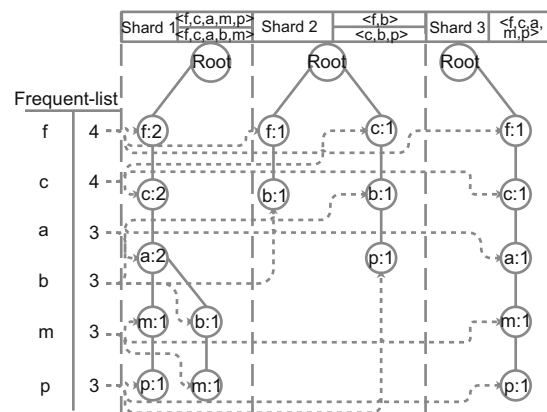
In the process of frequency counting, data are cut using the minimum threshold, which reduces the amount of data that must be processed, thereby improving processing efficiency. Upon completion of the frequent item count, we identify the values in descending order according to the frequency counting that matches the basic requirements of the FP algorithm. This part is needed to construct the FP-tree.

**4.2 Conditional pattern algorithm**

According to the FP algorithm theory, the conditional pattern is one of the main basic factors that are needed for FP mining. The local FP-tree is created, followed by the global F-list, which does not change the conditional pattern. In other words, the inter-relationship between itemsets will not be converted. Assuming that the data distribution is as shown in Table 2, through the F-list count algorithm, we can easily obtain the F-list:  $\langle f:4, c:4, a:3, b:3, m:3, p:3 \rangle$ .

The local frequent count should not affect the item order, because the global F-list already conforms to the item order in descending order. For example, in shard 2, the local frequency count for item “b” is two, and its global frequency count is three. However, the local frequency count for item “f”

is one, but the global frequency count is four. Based on the rule of the FP-tree, which does not consider the global F-list, item “b” should be in front of item “f” in descending order. This means that item “b” is the conditional pattern of item “f,” which does not conform to the global data situation, thus damaging the relationship between itemsets. Therefore, the process of creating the local FP-tree should follow the global F-list item order. The local FP-trees for the three shards are shown in Fig. 5. In shard 2, the conditional patterns of item “b” are  $\langle f \rangle$  and  $\langle c \rangle$ , which are the same as in the global data situation shown in Fig. 1.



**Fig. 5** Local FP-trees for Table 2

The conditional pattern algorithm is based on the reversed order of FP-growth, that is, according to the F-list bottom-up mining. The first item is “p,” followed by “m,” “b,” and so on, until item “f.” We consider item “p” as an example. After generating the FP-tree, we obtain the conditional patterns of item “p,” which are  $\langle f:1, c:1, a:1, m:1 \rangle$ ,  $\langle c:1, b:1 \rangle$ , and  $\langle f:1, c:1, a:1, m:1 \rangle$ . The conditional pattern of each shard can be incorporated in the reducing stage. This allows us to obtain the provisional conditional patterns of item “p,” which are given by  $\langle f:2, c:3, a:2, m:2 \rangle$ . The conditional pattern algorithm is described in Algorithm 2.

At this stage, we obtain the provisional conditional patterns because we do not abridge the infrequent items in the conditional pattern. Nevertheless, we are not prepared to perform pruning at this stage, because we assume that there are many machines that can be used in the reduction process in the big data environment. Using the shard ID as the key in both the mapping and reducing stages,

the conditional patterns of the same items can be in different machines performing the reduction process. In other words, the provisional conditional patterns of an item are not integrated in the global transition dataset. If we cut the infrequent item out of the provisional conditional patterns, we can lose the opportunity to identify the FP or obtain the incorrect results. Therefore, simplification and the complete FP mining process are performed in the next phase.

---

**Algorithm 2** Conditional pattern algorithm
 

---

**Procedure:**

- 1: Mapper(key, value =  $T_i$ )
- 2: Load Shard\_list
- 3: **for** each  $T_i$  in Shard\_ID **do**
- 4:   Call Output(< Shard\_ID,  $T_i$  >)
- 5: **end for**

**Procedure:**

- 1: Reduce(key = Shard\_ID, value =  $DB_{Local}$ )
  - 2: Load F\_list
  - 3:  $FPtree_{Local} \leftarrow$  clear
  - 4: Define and clear conditional pattern: CP
  - 5: **for** each  $T_i$  in  $DB_{Local}$  **do**
  - 6:   Call CreateFPtree( $FPtree_{Local}, T_i, F\_list$ )
  - 7: **end for**
  - 8: **for** each  $a_i$  in  $FPtree_{Local}$  **do**
  - 9:    $CP_{a_i} \leftarrow$  CreateCP( $FPtree_{Local}, a_i$ )
  - 10:   Call Output(< null,  $a_i + CP_{a_i}$  >)
  - 11: **end for**
- 

### 4.3 Complete frequent pattern algorithm

Frequent patterns and conditional patterns exist in the following relationships:

$$\begin{aligned} & \text{Frequent\_Pattern} \\ &= \sum (\text{Conditional\_Pattern} \cup \text{Frequent\_Item}). \end{aligned} \quad (1)$$

In Section 4.2, we obtained the provisional conditional patterns using the MapReduce model. However, we also need to obtain the complete FP by further processing the temporary conditional patterns. The process is divided mainly into two stages: the first stage is incorporating all the temporary conditional patterns and then deleting the infrequent items in the temporary conditional patterns using the minimum threshold, and the second stage is using the final conditional pattern and descending order in the F-list to revive the complete FP for each itemset.

Discovering the final conditional pattern and mining the complete FP needs to use the MapReduce model once. Therein, the item is the key and the temporary conditional pattern is the value for each key. In this way, we ensure that the temporary conditional patterns of the same item are distributed into the same reduction process. We consider item “m,” although item “m” is processed in a different mapping stage, because the key is equal to the item, all the temporary conditional patterns (<f:1, c:1, a:1, b:1>, <f:1, c:1, a:1>, and <f:1, c:1, a:1>) of item “m” will be distributed in the same reducing stage (Fig. 6). The global temporary conditional pattern of item “m” is <f:3, c:3, a:3, b:1>, which is achieved by combining <f:1, c:1, a:1, b:1>, <f:1, c:1, a:1>, and <f:1, c:1, a:1>. Owing to <f:3, c:3, a:3, b:1>, we can obtain the global temporary conditional pattern for item “m,” and then delete the infrequent item <b:1> using the minimum threshold. In doing so, we obtain the final conditional pattern. Without the deletion step, a problem similar to that in the Apriori algorithm will appear in the process of mining, thus resulting in a large number of candidate sets.

Therefore, we can achieve the complete FPs using the recursive method, which can then be used to obtain the final conditional patterns and descending order of each item. We continue taking item “m” as an example. The final conditional pattern base of “m” is <f:3, c:3, a:3>, from the global FP-tree or all the local FP-tree by item “m” (Fig. 7). The complete FP mining process of item “m” is shown in Fig. 8. The FP mining by item “m” is the descending order from “f” to “m” (f←c←a←m). Hence, following Eq. (1), we can obtain the 1D FPs of item “m,” which are <am:3>, <cm:3>, and <fm:3>. Next, we consider obtaining the conditional pattern bases of two-dimensional (2D) items (“am,” “cm,” and “fm”) and the FPs. The conditional pattern base of “fm” is null; thus, we keep computing for “am” and “cm,” whose FPs are <cam:3>, <fam:3>, and <fcm:3>. In the same way, the conditional pattern bases of “fcm” and “fam” are null; we just have to think about the conditional pattern base of “cam.” Finally, the FP of the three-dimensional (3D) “cam,” which is <fcam:3>, is achieved. We also obtain the complete FPs of item “m,” which are <am:3>, <cm:3>, <fm:3>, <cam:3>, <fam:3>, <fcm:3>, and <fcam:3>. The complete FP algorithm is described in Algorithm 3.



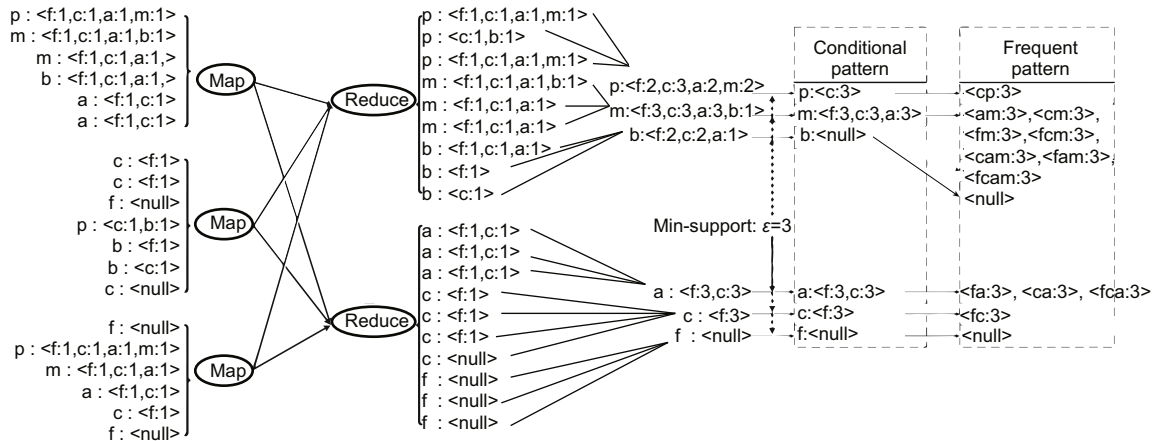


Fig. 6 Complete frequent pattern mining by the MapReduce process

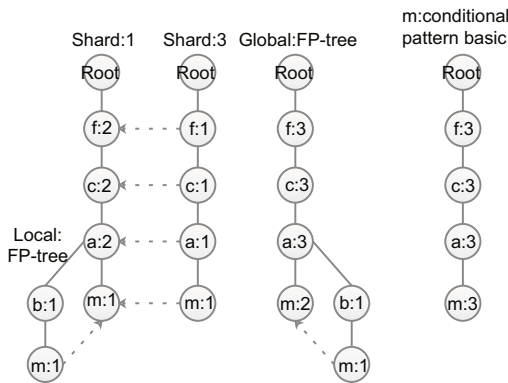


Fig. 7 Generating the conditional pattern base for  $m$

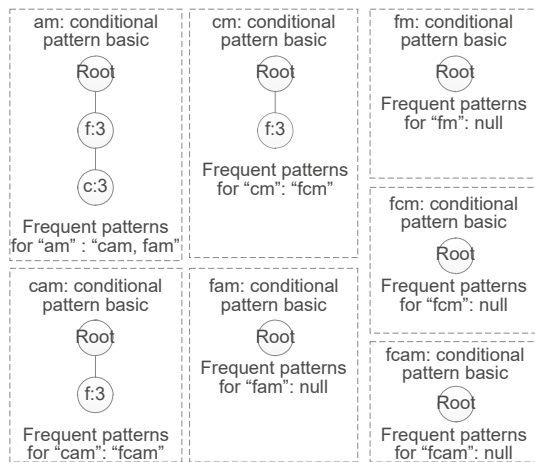


Fig. 8 Process of complete FP mining for  $m$

The NG-PFP algorithm can determine the complete FPs using the MapReduce model. Dividing by the task size and reversing the traditional grouping model can solve the problem of data redundancy and reduce the overhead of data migration.

### Algorithm 3 Complete frequent pattern algorithm

**Procedure:**

- 1: Mapper(key, value =  $a + CP_a$ )
- 2: for each  $a_i$  in  $a$  do
- 3: Call Output( $\langle a_i, CP_{a_i} \rangle$ )
- 4: end for

**Procedure:**

- 1: Reduce(key =  $a_i$ , value = Merge( $CP_{a_i}$ ))
- 2: Define and clear complete frequent pattern: CFP
- 3: min\_support  $\leftarrow \epsilon$
- 4: for each  $a_i$  in Merge( $CP_{a_i}$ ) do
- 5: for each item in  $CP_{a_i}$  do
- 6: if Count  $\geq$  min\_support then
- 7: Continue
- 8: else
- 9: Delete item from  $CP_{a_i}$
- 10: end if
- 11: end for
- 12: end for
- 13: for each  $a_i$  in  $CP_{a_i}$  do
- 14: CFP $_{a_i} \leftarrow$  CreateCFP( $CP_{a_i}, a_i$ )
- 15: Call Output( $\langle \text{null}, a_i + CFP_{a_i} \rangle$ )
- 16: end for

## 5 Evaluation

Three factors must be considered in performing data mining in a big data environment, i.e., correctness of the results, time validity required by the computing process, and the cost of computing space. We used nine machines with Intel Core i5-4200M QuadCore 2.5 GHz CPU and 8.00 GB RAM. All the experiments were performed on the Ubuntu

14.04 OS with Hadoop 2.0 and JDK 1.7.0 for performing the related algorithm, one master node, and eight slave nodes. Because of the smaller cluster test environment, we used WebDocs, a real-life huge transactional dataset (Lucchese et al., 2004), as the test dataset for evaluating the computing space occupancy rate and computational efficiency of the PFP, improved PFP (IPFP), and NG-PFP algorithms.

First, we considered the correctness of the results. Both the PFP and NG-PFP algorithms used the FP-growth method (Han et al., 2000) for frequent pattern mining. PFP is generally mined according to conditional pattern bases and the global FP-tree, while the NG-PFP algorithm adopts all local FP-trees contained in the mining item. For the mining item, the paths by the global FP-trees and all the local FP-tree of this item should be consistent, so we obtained the same conditional pattern bases. In the experimental results, both the PFP and NG-PFP algorithms obtained consistent and correct frequency patterns.

Then we tested the PFP and IPFP algorithms with an increase in the number of groups. The computing space occupancy rate gradually increased and the computational efficiency was also affected. In this study, the PFP and IPFP algorithms strongly depend on the group list. We found that the performances of the PFP and IPFP algorithms were affected by the number of groups. The maximum cost occurred when the number of groups is equal to the number of itemsets. However, for the NG-PFP algorithm, the results of the computing space utilization rate and computational efficiency were consistent (Figs. 9 and 10). Hence, compared with the PFP and IPFP algorithms, the computation cost of the NG-PFP algorithm is relatively manageable.

Next, we examined the shard datasets and observed the increase in the amount of data, computing space occupancy rate, and computational efficiency of the NG-PFP, PFP, and IPFP algorithms. The results for the three algorithms are shown in Figs. 11 and 12. The NG-PFP algorithm significantly improved space utilization in the second phase of the MapReduce process but increased the amount of temporary data in the third phase. Overall, the NG-PFP algorithm can improve the computing space occupancy rate by 32%–37%. The main reason lies in three aspects: distribution of the data itself, the number of items, and the length of the transaction

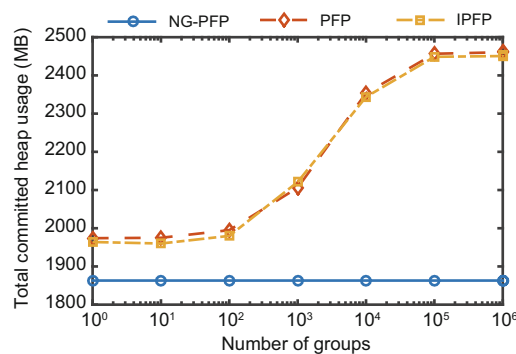


Fig. 9 Computing space occupancy rate

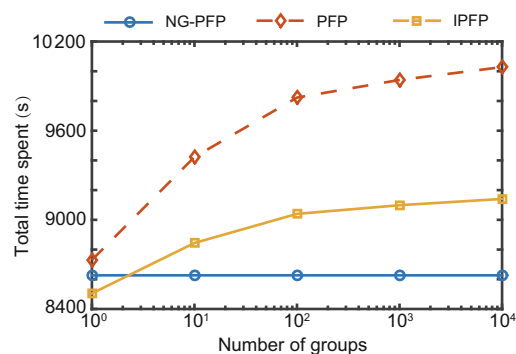


Fig. 10 Total time spent by groups

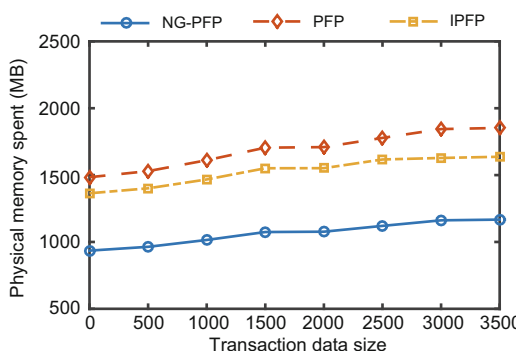


Fig. 11 Physical memory spent by data

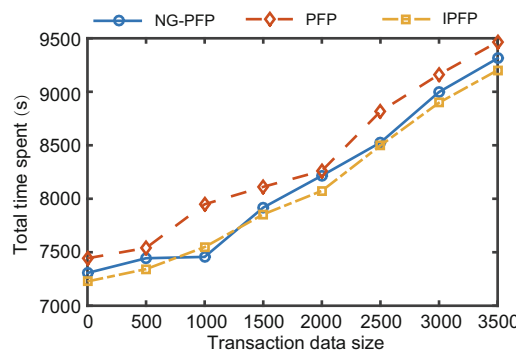


Fig. 12 Total time spent by data

dataset. In other words, data distribution is more spread out if the number of items is higher and the length of transaction data is longer. Hence, the

NG-PFP algorithm is more advantageous than the PFP algorithm. On the other hand, the gap in the computing efficiency between these algorithms is not very obvious. The main reason for this may be the presence of relatively small datasets, in which the cluster distributions are relatively concentrated. However, we believe that the NG-PFP algorithm is reduced by the grouping and the group of data migration is reduced, which improves computing efficiency to some extent.

## 6 Conclusions

In this study, we have analyzed the main problem of PFP algorithm frequent item omission and the main reason for data redundancy caused by grouping in the PFP algorithm. The solution process of complete frequent itemsets in the FP algorithm was to find the conditional pattern bases of the itemsets. To do so, we proposed an NG-PFP algorithm based on the conditional pattern bases. This algorithm controls the sequential structure of the local FP-tree using the global F-list, and therefore obtains the conditional pattern bases of all frequent items using the MapReduce model. The algorithm then achieved dimension reduction against conditional pattern bases through the minimum threshold and obtained the complete frequent itemsets. Using the proposed method, the item grouping process can be avoided, and the data redundancy caused by multiple allocations was reduced. The NG-PFP algorithm, based on the conditional pattern bases, also reduces the required computing space and the consumption cost of parallel processing associated with data mining. In future work, we aim to study the distribution of complex data clustering for further verification.

### Compliance with ethics guidelines

Zhe-jun KUANG, Hang ZHOU, Dong-dai ZHOU, Jin-peng ZHOU, and Kun YANG declare that they have no conflict of interest.

### References

- Agrawal R, Srikant R, 1994. Fast algorithms for mining association rules in large databases. *Proc Int Conf on Very Large Data Bases*, p.487-499.
- Agarwal RC, Aggarwal CC, Prasad VVV, 2002. A tree projection algorithm for generation of frequent item sets. *J Parallel Distrib Comput*, 61(3):350-371. <https://doi.org/10.1006/jpdc.2000.1693>
- Bauer M, Bruveris M, Charon N, et al., 2018. A relaxed approach for curve matching with elastic metrics. <https://arxiv.org/abs/1803.10893>
- Berti-Équille L, Harmouch H, Naumann F, et al., 2018. Discovery of genuine functional dependencies from relational data with missing values. *Proc VLDB Endowment*, 11(8):880-892. <https://doi.org/10.14778/3204028.3204032>
- Caruccio L, Deufemia V, Polese G, 2016. On the discovery of relaxed functional dependencies. *Proc 20th Int Database Engineering & Applications Symp*, p.53-61. <https://doi.org/10.1145/2938503.2938519>
- Caruccio L, Deufemia V, Polese G, 2017. Evolutionary mining of relaxed dependencies from big data collections. *Proc 7th Int Conf on Web Intelligence, Mining and Semantics*, Article 5. <https://doi.org/10.1145/3102254.3102259>
- Caruccio L, Polese G, Tortora G, 2018. Dependency-based query/view synchronization upon schema evolutions. *Int Conf on Conceptual Modeling*, p.91-105. [https://doi.org/10.1007/978-3-030-01391-2\\_17](https://doi.org/10.1007/978-3-030-01391-2_17)
- Chen JC, Chen YG, Du XY, et al., 2013. Big data challenge: a data management perspective. *Front Comput Sci*, 7(2):157-164. <https://doi.org/10.1007/s11704-013-3903-7>
- Cong S, Han J, Padua D, 2005. Parallel mining of closed sequential patterns. *Proc 11th ACM SIGKDD Int Conf on Knowledge Discovery in Data Mining*, p.562-567.
- Deng LL, Lou YS, 2015. Improvement and research of FP-growth algorithm based on distributed spark. *Proc Int Conf on Cloud Computing and Big Data*, p.105-108. <https://doi.org/10.1109/CCBD.2015.15>
- di-Jorio L, Laurent A, Teisseire M, 2009. Mining frequent gradual itemsets from large databases. *Int Symp on Intelligent Data Analysis*, p.297-308. [https://doi.org/10.1007/978-3-642-03915-7\\_26](https://doi.org/10.1007/978-3-642-03915-7_26)
- El-Hajj M, Zaïane OR, 2006. Parallel bifold: large-scale parallel pattern mining with constraints. *Distrib Parallel Datab*, 20(3):225-243. <https://doi.org/10.1007/s10619-006-0445-0>
- Ge KS, Su HY, Li DS, et al., 2017. Efficient parallel implementation of a density peaks clustering algorithm on graphics processing unit. *Front Inform Technol Electron Eng*, 18(7):915-927. <https://doi.org/10.1631/FITEE.1601786>
- Han JW, Pei J, Yin YW, 2000. Mining frequent patterns without candidate generation. *ACM SIGMOD Rec*, 29(2):1-12. <https://doi.org/10.1145/335191.335372>
- Huhtala Y, Kärkkäinen J, Porkka P, et al., 1999. Tane: an efficient algorithm for discovering functional and approximate dependencies. *Comput J*, 42(2):100-111. <https://doi.org/10.1093/comjnl/42.2.100>
- Kruse S, Naumann F, 2018. Efficient discovery of approximate dependencies. *Proc VLDB Endowment*, 11(7):759-772. <https://doi.org/10.14778/3192965.3192968>
- Li HY, Wang Y, Zhang D, et al., 2008. PFP: parallel FP-growth for query recommendation. *Proc ACM Conf on Recommender Systems*, p.107-114. <https://doi.org/10.1145/1454008.1454027>
- Li N, Zeng L, He Q, et al., 2012. Parallel implementation of Apriori algorithm based on MapReduce. *Proc 13th ACIS Int Conf on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing*, p.236-241. <https://doi.org/10.1109/SNPD.2012.31>

- Lin KW, Chung SH, 2015. A fast and resource efficient mining algorithm for discovering frequent patterns in distributed computing environments. *Fut Gener Comput Syst*, 52:49-58.  
<https://doi.org/10.1016/j.future.2015.05.009>
- Lin MY, Lee PY, Hsueh SC, 2012. Apriori-based frequent itemset mining algorithms on MapReduce. *Proc 6<sup>th</sup> Int Conf on Ubiquitous Information Management and Communication*, Article 26.  
<https://doi.org/10.1145/2184751.2184842>
- Liu JQ, Wu YS, Zhou QF, et al., 2015. Parallel Eclat for opportunistic mining of frequent itemsets. *Int Conf on Database and Expert Systems Applications*, p.401-415.  
[https://doi.org/10.1007/978-3-319-22849-5\\_27](https://doi.org/10.1007/978-3-319-22849-5_27)
- Lucchese C, Orlando S, Perego R, et al., 2004. WebDocs: a real-life huge transactional dataset. *Proc IEEE ICDM Workshop on Frequent Itemset Mining Implementations*.
- Mandros P, Boley M, Vreeken J, 2017. Discovering reliable approximate functional dependencies. *Proc 23<sup>rd</sup> ACM SIGKDD Int Conf on Knowledge Discovery and Data Mining*, p.355-363.  
<https://doi.org/10.1145/3097983.3098062>
- Riondato M, DeBrabant JA, Fonseca R, et al., 2012. PARMA: a parallel randomized algorithm for approximate association rules mining in MapReduce. *Proc 21<sup>st</sup> ACM Int Conf on Information and Knowledge Management*, p.85-94.  
<https://doi.org/10.1145/2396761.2396776>
- Siddiqi A, Karim A, Gani A, 2017. Big data storage technologies: a survey. *Front Inform Technol Electron Eng*, 18(8):1040-1070.  
<https://doi.org/10.1631/FITEE.1500441>
- Srikant R, Agrawal R, 1996. Mining sequential patterns: generalizations and performance improvements. *Int Conf on Extending Database Technology*, p.1-17.  
<https://doi.org/10.1007/BFb0014140>
- Wang F, Hu L, Zhou J, et al., 2015. A survey from the perspective of evolutionary process in the Internet of Things. *Int J Distrib Sen Networks*, 11(3):462752.  
<https://doi.org/10.1155/2015/462752>
- Wang J, Han J, 2004. BIDE: efficient mining of frequent closed sequences. *Proc 20<sup>th</sup> Int Conf on Data Engineering*, p.79-90.  
<https://doi.org/10.1109/ICDE.2004.1319986>
- Xia D, Zhou Y, Rong Z, et al., 2013. IPFP: an improved parallel FP-growth algorithm for frequent itemsets mining. *Proc 59<sup>th</sup> ISI World Statistics Congress*, p.4034-4039.
- Xia D, Rong Z, Zhou Y, 2014. A novel parallel algorithm for frequent itemsets mining in massive small files datasets. *ICIC Expr Lett Part B*, 5(2):459-466.
- Yang Q, Du FY, Zhu X, et al., 2016. Improved balanced parallel FP-growth with MapReduce. *Joint Int Conf on Artificial Intelligence and Computer Engineering and Int Conf on Network and Communication Security*, p.1-5.  
<https://doi.org/10.12783/dtcese/aice-ncs2016/5681>
- Yang XY, Liu Z, Fu Y, 2010. MapReduce as a programming model for association rules algorithm on Hadoop. *Proc 3<sup>rd</sup> Int Conf on Information Sciences and Interaction Sciences*, p.99-102.  
<https://doi.org/10.1109/ICICIS.2010.5534718>
- Yu KM, Zhou JY, Hsiao WC, 2007. Load balancing approach parallel algorithm for frequent pattern mining. *Proc Int Conf on Parallel Computing Technologies*, p.623-631.  
[https://doi.org/10.1007/978-3-540-73940-1\\_63](https://doi.org/10.1007/978-3-540-73940-1_63)
- Zaki MJ, 2000. Scalable algorithms for association mining. *IEEE Trans Know Data Eng*, 12(3):372-390.  
<https://doi.org/10.1109/69.846291>
- Zaki MJ, 2001a. Parallel sequence mining on shared-memory machines. *J Parallel Distrib Comput*, 61(3):401-426.
- Zaki MJ, 2001b. SPADE: an efficient algorithm for mining frequent sequences. *Mach Learn*, 42(1-2):31-60.  
<https://doi.org/10.1023/A:1007652502315>
- Zhang XL, Breitinger F, Baggili I, 2016. Rapid Android parser for investigating DEX files (RAPID). *Dig Investig*, 17:28-39.  
<https://doi.org/10.1016/j.diin.2016.03.002>
- Zhang XL, Baggili I, Breitinger F, 2017. Breaking into the vault: privacy, security and forensic analysis of Android vault applications. *Comput Secur*, 70:516-531.  
<https://doi.org/10.1016/j.cose.2017.07.011>
- Zhang ZG, Ji GL, Tang MM, 2013. MREclat: an algorithm for parallel mining frequent itemsets. *Proc Int Conf on Advanced Cloud and Big Data*, p.177-180.  
<https://doi.org/10.1109/CBD.2013.22>
- Zhao YX, Zhang WX, Li DS, et al., 2016. Pegasus: a distributed and load-balancing fingerprint identification system. *Front Inform Technol Electron Eng*, 17(8):766-780.  
<https://doi.org/10.1631/FITEE.1500487>
- Zheng XF, Wang S, 2014. Study on the method of road transport management information data mining based on pruning Eclat algorithm and MapReduce. *Proc Soc Behav Sci*, 138:757-766.  
<https://doi.org/10.1016/j.sbspro.2014.07.254>
- Zhou L, Zhong ZY, Chang J, et al., 2010. Balanced parallel FP-growth with MapReduce. *Proc. IEEE Youth Conf on Information, Computing and Telecommunications*, p.243-246.  
<https://doi.org/10.1109/YCICT.2010.5713090>
- Zhuang YT, Wu F, Chen C, et al., 2017. Challenges and opportunities: from big data to knowledge in AI 2.0. *Front Inform Technol Electron Eng*, 18(1):3-14.  
<https://doi.org/10.1631/FITEE.1601883>