Soft Comput
DOI 10.1007/s00500-015-1930-z

METHODOLOGIES AND APPLICATION

A parallel algorithm for mining constrained frequent patterns using MapReduce

Xiaowu Yan[1] · **Jifu Zhang**[1] · **Yaling Xun**[1] · **Xiao Qin**[2]

Abstract Constrained frequent pattern refers to a frequent pattern generated using constrained conditions given by users and has characteristics of stronger pertinence, higher practicability and mining efficiency, etc. With the increasing of datasets, there are defects during the construction of the constrained frequent pattern tree, so that the constrained frequent pattern tree is difficult to apply to massive datasets. In this paper, a parallel mining algorithm of the constrained frequent pattern, called PACFP, is proposed using the MapReduce programming model. First, key steps in the algorithm, such as mapping transaction in datasets to frequent item support count, constructing the constrained frequent pattern tree, generating the constrained frequent pattern, and aggregating frequent patterns, are implemented by three pairs of Map and Reduce functions. Second, migration of data recording is achieved by applying a data grouping strategy based on frequent item support, and load balance is effectively solved while generating the constrained frequent pattern. In the end, experimental results validate availability, scalability, and expandability of the algorithm using celestial spectrum datasets.

Keywords Association rule · Constrained frequent pattern · MapReduce · Frequent item support · Load balance

Communicated by V. Loia.

✉ Jifu Zhang
jifuzh@sina.com

[1] School of Computer Science and Technology, Taiyuan University of Science and Technology, Taiyuan 030024, China

[2] Department of Science and Software Engineering, Auburn University, Auburn, AL 36849-5347, USA

1 Introduction

With the rapid increasing of datasets, how to store, process, and analyze datasets has become an important research topic. Association rule mining, which is an important branch of data mining (Chen et al. 1996), is to find inter-dependencies and association relationships among multiple things, then predict and discover one of the things via other things (Han et al. 2006; Agrawal et al. 1993). Mining efficiency is the key for extracting the association rules from massive and high-dimensional datasets because of large calculation and high I/O cost. MapReduce (Dean et al. 2008; Yang et al. 2010), which is presented by Google, is a simple, parallel, and distributed programming model. Hadoop platform (White et al. 2012; Lam et al. 2010) is a Java open source implementation of MapReduce. The Hadoop simplifies the programming of parallel computing, and it is widely used to process and analyze the massive datasets on the cluster environment. With the reliable storage and powerful computing ability of Hadoop, research and development of the parallel data mining algorithm based on Hadoop platform is an effective way to improve the efficiency and reduce I/O cost for association rule mining.

The generation of frequent patterns is a key step affecting the efficiency of association rules mining (Agrawal et al. 1993). The constrained frequent pattern is one of the frequent patterns and satisfies certain conditions that are the long-term accumulated experience and interests, and in-depth understanding of datasets in special areas, so that the effectiveness, pertinence, and practicability of the association rule mining can be effectively improved (Zhang et al. 2013). However, for massive and high-dimensional datasets, large memory consumption, high I/O cost, and high time and space complexity have become the key problems of the constrained frequent patterns mining. In this paper, we present a paral-

Published online: 17 November 2015

lel constrained frequent pattern mining algorithm on Hadoop platform based on the in-depth analysis of constrained frequent pattern mining algorithms. First, three pairs of Map and Reduce functions are used to implement the whole process including mapping transactions to frequent items support counting, constructing constrained frequent pattern tree (CFP-Tree), mining frequent patterns, and aggregating frequent patterns results. Second, a load balancing strategy is proposed by reallocating the data records according to the frequent items support to effectively solve the load imbalance problem. Finally, the experimental results validate the efficiency, scalability, and extensibility of our algorithm.

In this study, the contributions and findings are summarized as follows:

- We propose a parallel mining procedure and method of the constrained frequent pattern mining algorithm.
- A parallel mining algorithm of the constrained frequent pattern, called PACFP, is proposed using the MapReduce programming model.
- We propose a load balancing strategy based on frequent item support.
- Experiments validate availability of the algorithm and the load balancing strategy using celestial spectrum datasets.

2 Related work

The association rule, which was first proposed by Agrawal (1993), is an important research topic to objectively reflect the correlation or association relationship among items of datasets. An important step of association rule mining is to discover the frequent patterns (Han et al. 2006; Agrawal et al. 1993). The classic frequent pattern generation algorithms can be mainly divided into two categories: Apriori algorithm (Agrawal et al. 1993) and FP-Growth algorithm (Han et al. 2000, 2004). In Apriori algorithm, the iteration search method is used to generate frequent patterns step by step. However, the inherent defects of Apriori algorithm are as follows:

- The high I/O cost and time-and-space overhead are caused by multiple scanning of the database repeatedly.
- A large amount of candidate itemsets are generated.
- The efficiency of algorithm will be significantly decreased with an increasing number of items in the frequent itemsets.

2.1 FP-growth algorithm of mining frequent patterns

FP-Growth algorithm (Han et al. 2000, 2004)—another classic algorithm for mining frequent patterns adopts a compressed storage data structure called FP-Tree (frequent pattern tree). It significantly improves the efficiency of generating frequent patterns by scanning the database only twice and not generating the conditional itemsets. Existing typical studies on FP-Growth algorithm are shown as follows. FPIFM algorithm is proposed in Gao et al. (2010), which overcomes the shortcomings of repeatedly reading the same branches when traversing the FP-Tree in traditional FP-Growth algorithm to get frequent patterns. In the algorithm, the node domain of each node stored all its predecessor nodes and then computed sub-conditional pattern bases and generated frequent patterns. An efficient algorithm for mining maximum frequent patterns based on FP-Tree named IAFP-Max, which is proposed in Wang et al. (2010), reduces the memory consumption in the mining process, and it improves the efficiency of the algorithm by introducing the concept of suffix sub-tree. To reduce the scale of constructing FP-Tree, save traverse time, and improve the efficiency, a maximum frequent itemsets mining algorithm (MMFI) based on FP-Tree is presented in Hui-ling et al. (2012), in which array and matrix are adopted to store datasets. An improved FP-Tree algorithm named MFI is proposed in Islam et al. (2011). MFI need not produce conditional FP-Tree to reduce memory usage, thus the efficiency of mining was significantly improved. An interrelation analysis method of celestial spectra data using constrained frequent pattern trees is proposed in Zhang et al. (2013). In the method, first-order predicate logic is used to represent knowledge derived from celestial spectra data, and the concept of constrained frequent pattern tree is presented to mining association rules, aiming to improve the efficiency and pertinence of association rule mining. At the same time, the CPU and I/O performance of the interrelation analysis method are evaluated quantitatively. Although extensive researches made lots of improvement on FP-Tree structure and FP-Growth algorithm, its natural problems have not been solved, including the following: the algorithms need to traverse conditional pattern bases in the process of generating new FP-Trees which will repeatedly apply for massive local or server data resources, thus efficiency of the algorithms and load of database server will be deteriorated. Since too much memory is occupied during constructing FP-tree, FP-growth algorithm cannot construct FP-Tree in the memory for the massive and high-dimensional datasets.

2.2 Distributed and parallel mining of frequent patterns

Distributed and parallel computation is a necessary tendency for enhancing the efficiency of association rules mining, can effectively improve the existing problems in the current frequent pattern mining algorithms, and is more suitable for analysis and mining of massive datasets. Typical research works of distributed and parallel frequent pattern mining are as follows. A parallel frequent itemsets mining PFP-Tree algorithm is proposed in Javed et al. (2004). In the algo-

rithm, results in having entire counting data structure do not duplicate on each processor and communication overheads are reduced by efficiently partitioning the list of frequent elements list over processors. However, PFP-Tree algorithm requires each node to exchange the conditional patterns bases based on each frequent 1-itemsets, so that the communication cost is relatively large and mining efficiency is low. Furthermore, the proposed algorithm introduces a parallel algorithm for mining association rules based on FP-tree, namely PAMARF in Tu et al. (2011). In the algorithm, database is split on horizontal and local frequent itemsets that are generated in each node, then global frequent patterns are obtained by exchanging data records under the control of a center node. Due to using top-down strategy, communication overhead is decreased and mining efficiency is improved. A parallel association rules algorithm created by combining FP-growth algorithm and the idea of parallel computing is put forward in Chen et al. (2011). In the algorithm, data partitioning, task allocation, and load balancing have been further studied to improve mining efficiency and make it suitable for massive data mining. In Zhou et al. (2008), a TPFP-tree structure is designed based on transaction ID, that is, Tidset-based Parallel FP-tree, which can directly choose transactions without scanning database in the transmission of transactions. In the algorithm, cost of communication and updating TPFP-tree is reduced and there is better scalability than the PFP-tree.

Googles MapReduce framework is known as the framework of Clouding Computing. With development and application of cloud computing, more and more researchers pay attention to algorithms based on MapReduce programming model (Rong et al. 2013; Li et al. 2008; Hong et al. 2013; Zhou et al. 2010; Seki et al. 2013; Chen et al. 2013; Liu et al. 2009), such as trajectory pattern mining algorithm in Seki et al. (2013), sequential pattern mining algorithm based on MapReduce model on the Cloud (abbreviated as SPAMC) in Chen et al. (2013), and MRPF in Liu et al. (2009) which used to discovery the prescription data. At the same time, many researchers implement the FP-Growth algorithm based on MapReduce programming model. Apriori algorithm and FP-Growth algorithm on MapReduce are implemented in Rong et al. (2013). Meanwhile, speedup ratio, scalability, and reliability of the algorithms are experimentally validated by selecting different sizes of datasets and different computing nodes. PFP (a parallel FP-Growth) algorithm based on the MapReduce programming model is proposed in Li et al. (2008), in which the execution tasks for various stages are described. In the algorithm, the mining task can be automatically decomposed into many independent computing tasks, and the tasks are mapped into the MapReduce model. An IFP-Growth (improved FP-Growth) algorithm is proposed in Hong et al. (2013), and it improves the efficiency by reducing the searching space of itemsets. At the same time, IFP-Growth algorithm is implemented on MapReduce (MR-IFP-Growth) which can improve the capability to deal with the massive datasets. Parallel FP-Growth algorithm using MapReduce programming modelPFP is implemented in Zhou et al. (2010), and BPFP algorithm is also presented by adding a load balancing feature to PFP algorithm. The parallelism of the algorithm is improved, and the communication overhead is reduced.

In short, frequent pattern mining based on FP-Tree still has some defects, such as only considering the optimization of FP-Tree structure and data storage but not considering the pertinence and availability of frequent pattern. In Zhang et al. (2013), constrained frequent pattern tree can be generated using user experience and interest as well as in-depth comprehension in a certain field as background knowledge. Thereby, the pertinence and generation efficiency of frequent patterns extracted from the constrained frequent pattern tree can be effectively enhanced. MapReduce, which is a popular parallel programming model, not only simplifies the programming complexity of distributed computing, but also can achieve the processing and analysis for massive datasets. The MapReduce programming model is adopted to realize constrained frequent pattern mining algorithm so that the mining efficiency of constrained frequent pattern can be effectively enhanced and I/O cost caused by scanning for massive and high- dimensional datasets can be reduced.

3 Preliminaries

3.1 Frequent patterns and CFP-tree

Let DB be a transaction database and $I = \{i_1, i_2, \ldots, i_n\}$ be a set of n transaction itemsets in DB. Each transaction T is a subset of I, that is $T \subseteq I$, which is identified by a unique TID in DB. According to Chen et al. (1996); Han et al. (2006); Agrawal et al. (1993); Zhang et al. (2013), the concepts and method of constrained frequent pattern tree constructions are described as follows.

For any itemsets $X \subseteq I$,the transaction T supports the itemsets X if $X \subseteq T$. The number of transactions which contain itemsets X is the support count of X in the DB. The percentage of the transactions in DB is the support of X, that is, $\text{support}(X) = \text{count}(T)/|DB|$. If $\text{support}(X) \geq S_{min}$, which is the minimum support threshold given by users, we refer to itemsets X as frequent in DB. And the number of items is called the length or dimension of frequent itemsets or patterns. The essence of frequent pattern mining is to extract all frequent itemsets that satisfy the minimum support threshold.

In Zhang et al. (2013), the constrained frequent pattern mining algorithm extracts constrained frequent patterns based on CFP-Growth algorithm with the background knowl-

edge given by the users. To generate frequent patterns, the algorithm just scans the database twice to build a constrained frequent pattern tree (CFP-Tree) using the background knowledge, then it traverses the tree with a bottom-up strategy. For background knowledge G in a certain field, a frequent pattern P is described as a path from the root node of a FP-Tree to a leaf node of the tree. P is a constrained frequent pattern if P can satisfy G, and the FP-tree is called a constrained frequent pattern tree (CFP-Tree) if all frequent patterns in the tree are constrained frequent patterns.

Constrained frequent pattern mining algorithm needs to preset minimum support threshold, background, and experience knowledge. Since any one constrained frequent pattern satisfies the background knowledge, the CFP-Tree built by the transactions with constraint conditions will contain the interested frequent pattern. Hence, the CFP-Tree can be constructed through traversing database DB twice according to the following steps.

1. First, all the frequent 1-itemsets and their supports are obtained by scanning the database for the first time. Then, these frequent 1-itemsets are ordered descendingly based on support, and frequent item table L is generated to save the results.
2. Create root node of the CFP-Tree and mark as "NULL".
3. For each transaction T in database DB, if T contains the interesting pattern, then turn to step 4, else directly scan the next transaction.
4. A new transaction T' is formed by rearranging the items in transaction T based on the order appeared in list L of frequent items; then, update the CFP-Tree according to the following three steps.
    1. Search for the longest prefix path that is matching with T' in the CFP-Tree;
    2. Increase the count of the node with the matching prefix by 1;
    3. Find out the mismatching suffix in T', choose the node of the last frequent item in the longest matching prefix as the root node, then create child nodes successively in the CFP-Tree and set the count to 1.

3.2 Data migration

The traditional parallel FP-Tree algorithm divides the database into continuous partitions, while distributed FP-Trees have mutual interdependence which will bring about frequent synchronization problems in the parallel implementation process. In Li et al. (2008), a kind of data migration strategy is presented for implementing FP-Tree. Assuming the number of computing nodes is P, the average value A (A is a positive integer and a top integral) is gotten by the number of items in frequent 1-itemsets divided by the number P (i.e., computing node), which is the time to traverse frequent 1-itemsets for each computing node. Then, a group list is obtained. Based on the group list, redistributing the transactions into the corresponding group can be achieved by the following 2 steps:

1. Use Hash function to group the items in frequent 1-itemsets and assign group number Group-ID.
2. Read transaction Ti and start positioning from the last item, if this item is contained in one group, then send this transaction to the corresponding group. Transactions containing multiple group items only need to be transferred once. Sending the transactions to a certain group is to construct FP-tree in this calculation node, so that frequent itemsets containing the group items can be mined. The frequent itemsets containing one or more items belong to this group. Because the item in transaction is sorted by support count from high to low, it generates the overall frequent pattern containing the group items in each calculation node.

4 Parallelization of constrained frequent pattern mining

4.1 Parallel mining procedure

The construction of constrained frequent pattern tree needs to scan the database twice. The first traverse is to produce frequent 1-itemsets and the second traverse is to construct CFP-Tree according to background knowledge given by users. All frequent patterns can be obtained by continual recursive mining on CFP-Tree. For massive high-dimensional datasets, CFP-Tree may be too large to stay in the memory for a single computing node, and I/O cost of constructing the tree is very high (Zhang et al. 2013). The research on parallelization of constrained frequent pattern mining algorithm has very important realistic value. Constrained frequent patterns refer to a frequent pattern generated by constrained conditions given by users. So what we need to solve is the parallel processing of the constrained conditions to ensure correctness and completeness of global constrained frequent patterns.

Let the number of computing nodes be N in the parallel/distributed computing environment. The database DB can be divided into N subsets, that is $\mathrm{DB}_1, \mathrm{DB}_2, \ldots, \mathrm{DB}_n$, and are, respectively, distributed to N nodes. For a minimum support threshold Smin, the background knowledge G, and any one pattern P in DB, P is a global constrained frequent pattern of DB if P can satisfy G and $\mathrm{support}(P) \geq S_{\min}$ and denoted as $G(P) = \text{true}$ and $\mathrm{support}(P/DB) \geq S_{\min}$. Similarly, P is a local constrained frequent pattern of DBi if $G(P) = \text{true}$ and $\mathrm{support}(P/\mathrm{DB}_i) \geq S_{\min}$ $(i = 1, 2, 3, \ldots, N)$.

Theorem 1 *Let P be any global constrained frequent pattern in DB, the minimum support threshold be* $S_{\min}$*, and the background knowledge be G. If P is a global constrained frequent pattern in DB, P is at least the local constrained frequent pattern in* DB_i*, that is,* $G(P) = \text{true}$ *and support*$(P/DB_i) \geq S_{\min}$ $(i = 1, 2, 3, \ldots, N)$.

Proof Let P be a global constrained frequent pattern in DB, $S(P/\text{DB}) = |P/DB| \,/\, |DB| \geq S_{\min}$ and $|P/DB| \geq |DB| * S_{\min}$. Through proof by contradiction, assuming that P is not a local constrained frequent pattern in any DB_i, then for $\forall i$ $(i = 1, 2, 3, \ldots, N)$, $S(P/\text{DB}_i) = |P/\text{DB}_i|/|\text{DB}_i| < S_{\min}$ and $|P/\text{DB}_i| < |\text{DB}_i| * S_{\min}$ will be true.

$$|P/\text{DB}| = |P/\text{DB}_1| + |P/\text{DB}_2| + |P/\text{DB}_3| + \ldots + |P/\text{DB}_n|$$
$$< |\text{DB}_1| * S_{\min} + |\text{DB}_2| * S_{\min} + |\text{DB}_3| * S_{\min} + \ldots + |\text{DB}_n| * S_{\min}$$
$$= (|\text{DB}_1| + |/DB_2| + |\text{DB}_3| + \ldots + |\text{DB}_n|) * S_{\min}$$
$$= |\text{DB}| * S_{\min}$$

Contradicting with the known, the assumption is invalid. Then, P must be a local constrained frequent pattern in DB_i. □

Theorem 1 indicates that the global constrained frequent pattern is at least the local constrained frequent pattern of some data subsets. P is certainly not the global constrained frequent pattern if pattern P is not the local constrained frequent pattern of any data subset. Therefore, the union $L = L_1 \cup L_2 \cup \ldots \cup L_n$ of partial constrained frequent pattern $L_i = \{P | G(P) = \text{true and } |P/\text{DB}_i| \geq S_{\min}\}$ on each data subset can be as the candidate global constrained frequent itemsets L, so the global constrained frequent patterns are generated.

According to Zhang et al. (2013), the implementation steps of the constrained frequent pattern mining algorithm are illustrated in Fig. 1. Frequent 1-itemsets are obtained by first traversing datasets DB, then the datasets are updated based on the constraint conditions, and CFP-Tree is constructed. Finally, CFP-Growth algorithm is called to generate constrained frequent patterns.

1. In generating frequent 1-itemsets by scanning datasets, calculating the support counts for all items will be needed, where support count is the number of occurrence in the whole datasets for itemsets. We averagely divide datasets into N data subsets. Obviously, support counts of 1-itemsets in each subset are calculated, and are summed up to get the global support counts. Thus, the process of generating frequent 1-itemsets can be paralleled.
2. When the datasets are updated with global support counts, the items that cannot meet minimal support are pruned, and the others are sorted according to descending order of the support count. Each data subset is updated by scanning the N data subsets divided in step 1, respectively. The updating of the whole datasets can be completed; therefore, the process can also be paralleled.
3. In the constrained frequent patters mining algorithm, scanning datasets and constructing CFP-Tree are based on memory. Each data subsets can be scanned to construct local CFP-Tree, and the local CFP-Tree is traversed from bottom to top by calling CFP-Growth to obtain local constrained frequent patterns. The local frequent patterns are aggregated to generate candidate global frequent patterns. According to Theorem 1, global constrained frequent patterns can be correctly obtained by scanning data subsets, respectively. Thus, the generation procedure of frequent patterns can be paralleled.

From the above, the parallelization implementation procedure of constrained frequent patterns mining algorithm is shown in Fig. 2.

4.2 Parallel mining procedure using MapReduce

MapReduce programming model (Dean et al. 2008; Yang et al. 2010) provides simple and strong interface for users. MapReduce is widely applied since it can meet high-performance and distributed calculation demand for users on the general cluster environment constructed by common computing nodes. The basic concept of MapReduce is originated from two basic operations in the function programming model: Map (projection) and Reduce (stipulation or aggregation). The programmers can define the Map and Reduce functions. The Map function receives data blocks to process and produce intermediate results. The intermediate results produced by Map need to be sorted before reduce calculation. Shuffle mechanism sends these intermediate results with different key values to corresponding nodes, respectively. The Reduce functions of each node receive the local intermediate results to conduct statute by iterative Reduce functions and produce the final results. Briefly, MapReduce programming model is to use the Map and Reduce function to establish one link from input to output, that is, Map realizes input data distribution, and Reduce realizes core algorithm and output results.

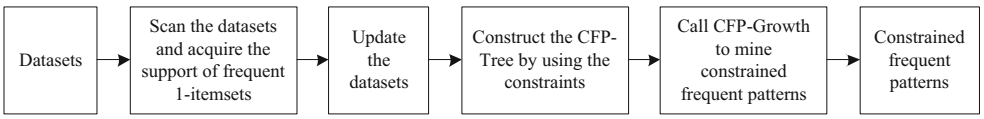

Fig. 1 The procedure of the constrained frequent pattern mining algorithm

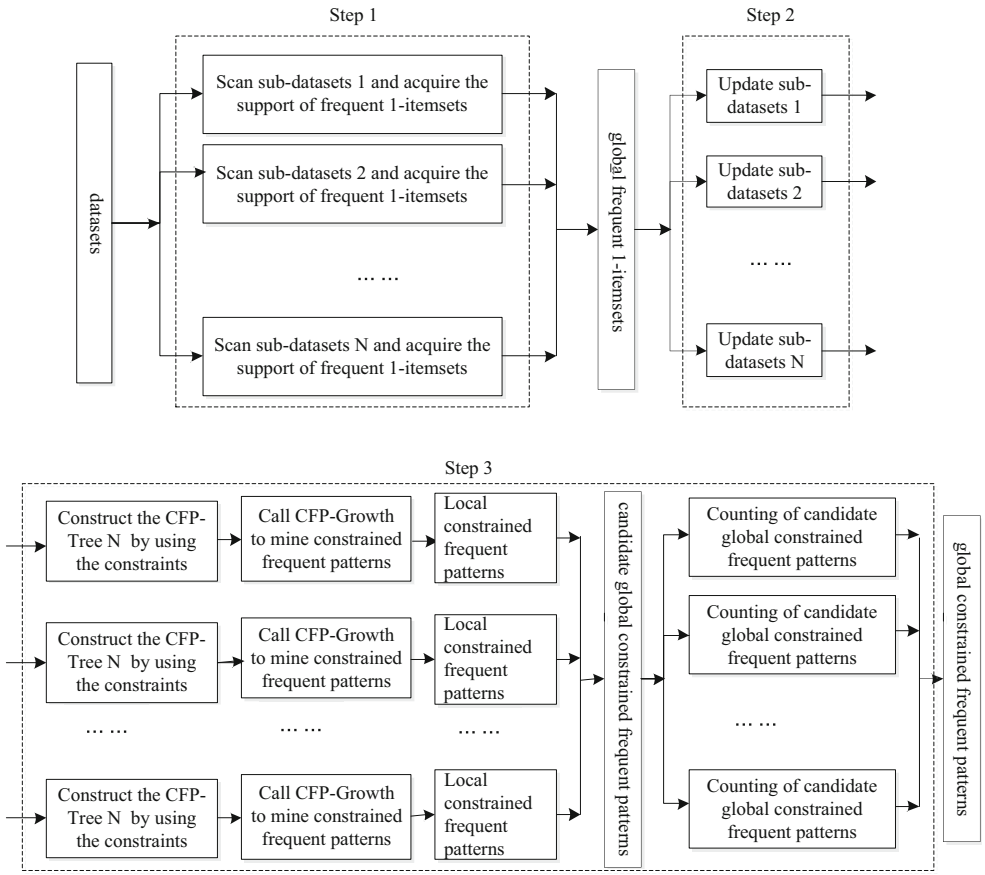

Fig. 2 The parallelization implementation procedure of constrained frequent pattern mining algorithm

Assuming the number of computing nodes in a cluster is N. According to Fig. 2, the parallel mining implementation steps of constrained frequent pattern are as follows under MapReduce programming model.

1. First, the datasets DB are decomposed into N continual partitions with the same size according to the number of transaction records, and each partition is distributed to different computing nodes and stored in the local disk.
2. For step 1 in Fig. 2, corresponding data partition is processed and the support count of all 1-itemsets is calculated in each computing node. According to the minimum support threshold, frequent 1-itemsets can be generated, and then the items of frequent 1-itemsets are sorted in L by descending support count order. This calculation process can be realized by one Map and one Reduce.
3. Step 2 and step 3 in Fig. 2 can be implemented by one Map and one Reduce, respectively.

Mapper The Mapper is mainly used to complete the repartition of datasets.

According to the obtained frequent 1-itemsets L, the local transactions are scanned and frequent items in the transactions are sorted by the order of L. Meanwhile, the patterns that users are not interested in are pruned, then the datasets DB are redistributed based on the data migration strategy in 3.2. The items belonging to the same new partition will be aggregated into one computing node, and form complete datasets to make preparation for constructing CFP-Tree and scheduling CFP-Growth to generate constrained frequent patterns on Reducer.

Reducer Each Reducer computing node constructs CFP-Tree and calls CFP-growth to generate constrained frequent patterns.

4. The final constrained frequent patterns can be obtained by aggregating the local frequent patterns generated in each computing node, which can be completed by one Map and one Reduce. In short, the parallel implement procedure of constrained frequent pattern mining and data structure are described in Fig. 3 under the MapReduce programming model. The procedure is decomposed into 3 pairs of Map and Reduce functions. First, we make a balanced partition on HDFS and calculate support counts of all the frequent items. Second, each computing node constructs CFP-Tree in parallelization and obtains constrained fre-

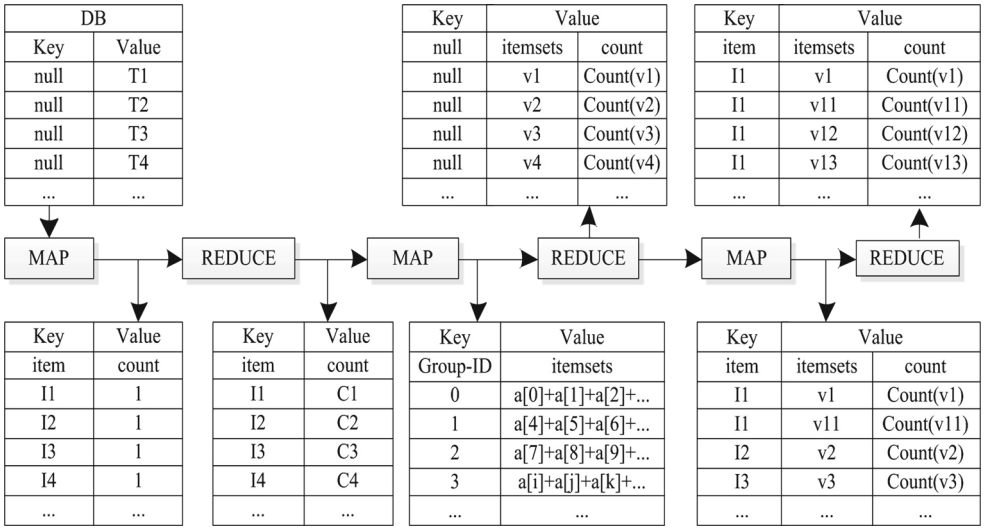

Fig. 3 The realization procedure based on MapReduce

quent patterns by traversing after data migration. Finally, all the patterns are aggregated to generate global frequent patterns.

5 Parallel mining algorithm of constrained frequent pattern

5.1 Parallel counting

Counting is a typical application of MapReduce, which is similar to the WordCount example provided by Hadoop. During task submission, the datasets are spitted into multiple blocks according to datasets in HDFS and the number of computing nodes in the cluster environment. The transactions are translated into key-value pairs by Map function and stored on Mapper nodes. Then, they are sent to corresponding Reducer nodes after being stored by shuffle mechanism. Finally, Reduce function makes sum and count the items with the same key to generate the final frequent 1-itemsets count.

Algorithm 1: parallel counting

```
Map Input: < key=null, value=T_i >
Map Process:
for each i in T_i do
  | Output < i,1 >;
end

Reduce Input: < i,1 >
Reduce Process:
C=0;
for each item i in T_i do
  | C= C+1;
end
```

Algorithm 2: parallel CFP-Growth

```
Map Input< key, value=T_i >
Map Process:
Load Group-list;
// Obtaining the Group-list;
Split( T_i )
// Traversing the database;
if item i in G then
    // i is an item in a transaction, and G is the background
    knowledge;
    a [ ] ← Split(T_i)
    for each j=| T_i |-1 to 0 do
        a[j] in group n
        Call output< n, a[0]+a[1]...+a[j] >;
    end
else
    Delete T_i;
end

Reduce Input: < key=GID, value=T'_i >
Reduce Process:
Load Group−list;
newGroup ← Group-list;
// newGroup is the transaction set assigen by Group-list in
the local calculated nodes; LocalCFP-tree ← clear;
for each T'_i in DB(GID) do
  | Call insert → build CFP-tree
end
for each a_i in Group-ID do
    Define and clear a size K max heap:k;
    CFPGrowth(LocalCFP-tree,ai,k);
end
for each v_i in k do
    Output(null,v_i+count(v_i)); // vi refers to the constrained
    frequent pattern
end
End
```

5.2 Parallel CFP-growth

This process is the core step of the algorithm. In the Map function, local datasets are scanned in each computing node

and infrequent items in the transactions based on the frequent 1-itemsets are pruned, and the transactions excluding the pattern that users take no interest in are deleted. Transactions are sorted in descending order of frequent item count and are sent to corresponding computing nodes based on the grouping strategy in Li et al. (2008). In the Reduce function, input of each computing node is the transaction set identified by Group-ID, and Local CFP-Trees are constructed. Meanwhile, Reduce function defines a stack with size K for frequent items in a local computing node, which is used to store constrained frequent items generated by CFP-Tree algorithm.

5.3 Result aggregation

Algorithm 3: result aggregation

```
Map Input: <key=null, value=v_i+count(v_i)>
Map Process:
for each item a_i in v_i do
    Output < a_i, v_i+count(v_i)>;
end

Reduce Input: <key=a_i, value=v_i+count(v_i)>
Reduce Process:
Define and clear a size K max heap: k;
for each v_i in v_i+count(v_i) do
    if k<K then
        insert v_i+count(v_i) into k;
    else
        Compare count(v_i) with item in k;
    end
    Output(a_i,C);
    // C is a set of highest frequent K constrained frequent
    patterns corresponging with a_i;
end
```

In this step, the global frequent patterns generated by the parallel CFP-growth in the previous step are ordered and stored in the stack by the frequency in the descending order, and the stack is updated when reading new constrained frequent patterns. The stack is guaranteed to store K constrained frequent patterns with the highest frequency during updating.

5.4 Dynamic load balance

Load balancing strategy is an effective way to enhance resource utilization and parallel processing performance in the cluster environment. The partition and distribution strategy of datasets will impact the load balance of a cluster system. In the homogeneous cluster system, the data partition only based on the number of frequent 1-itemsets will lead to the occurrences of frequent items having a great difference in each computing node, though the number of frequent 1-itemsets received is the same in the node. Therefore, the scale of CFP-Tree locally constructed is also different. If we just adopt static partition strategy, unbalanced load among processors will arise, and mining efficiency will drastically decrease.

Based on the data migration strategy in Sect. 3.2, the datasets need to be rescanned and transmit transactions to the corresponding data partition when constructing CFP-Tree and mining constrained frequent pattern. However, the unbalanced support count will lead the number of records sent to each node different that will cause the imbalance in communication of data transmission and the tree scale. To solve this problem, we first calculate the sum of support count of frequent 1-itemsets, and the sum is divided by the number of computing nodes N to acquire the average number of frequent 1-itemsets (represented as B). Then, we allocate the sorted frequent 1-itemsets to each computing node, in which the accumulative value of support count of frequent 1-itemsets on each computing node should not be greater than B. Multiple frequent items will be obtained in each node. This strategy will enable frequent 1-itemsets with greater support count to be distributed to more nodes, while frequent 1-itemsets with less support count will be combined and sent to one node, thereby, load balance will be effectively achieved.

In short, we propose the parallel mining algorithm PACFP of constrained frequent patterns under MapReduce programming model. Similar to Seki et al. (2013), the parallel generating constrained frequent pattern tree is equivalent to parallel generating frequent pattern tree when the constraint is null (no constraints).

6 Experimental evaluation

The experimental setup: 5 computing nodes constructed by virtual machine (512MB memory) on a server (8G memory) +11 computing nodes constructed by LENOVO desktop computers (PentiumIV 3.0 GHz processor, 512MB main memory, and the Linux operating system). Each node runs on the Ubuntu Linux 10.10 operating system with JDK 1.6.0-37, Hadoop-1.1.1, java eclipse, Hadoop-eclipse, etc. All the nodes in the cluster communicate with each other via SSH protocol.

The experimental datasets are the SDSS star spectrum data provided by the National Observatory of China. Spectral data are series of continuous data made up of different flux to which each wavelength is corresponding and physico-chemical properties of this spectrum. To intuitively describe the spectral data characteristics and physico-chemical properties, and better adapt to association rules mining, the celestial spectra datasets are discretized in Zhang et al. (2013). Each spectral data have 206 attributions, that is, the first 200 attributions are wavelength property and the last 6 attributions are the physico-chemical properties including temperature, luminosity, chemical abundance, micro-turbulence, and

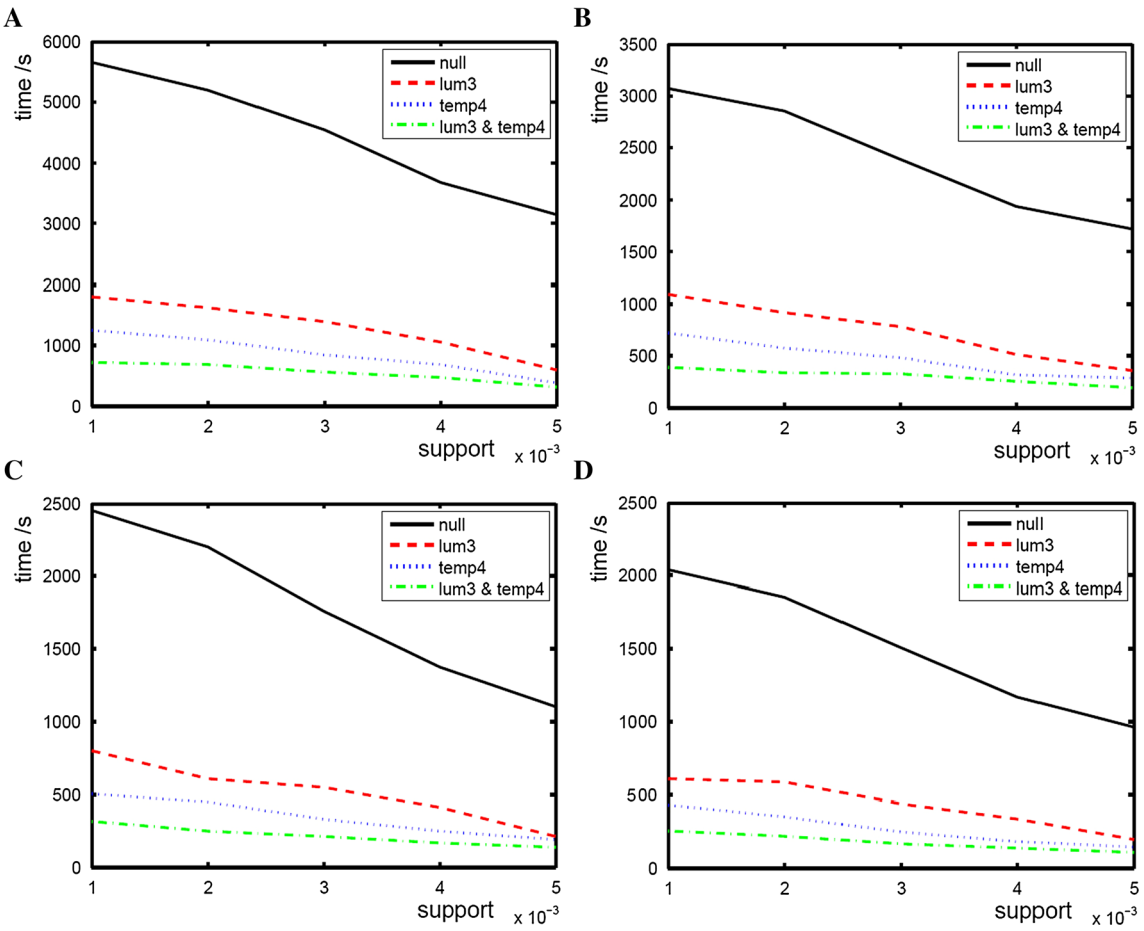

Fig. 4 The impact of support threshold on the mining efficiency (datasets = 2G) (The 4 subfigures are **a** node = 2; **b** node = 4; **c** node = 6; and **d** node = 8 in sequence)

other properties. In the following experiments, the different constraints can express as unll, lum3, temp4 and lum3 & temp4.

6.1 Minimum support

In the parallel constrained frequent pattern mining algorithm, support reflects the importance of frequent items. At the same time, the essence of mining constrained frequent patterns is to find the constrained frequent patterns which satisfy the minimum support threshold and background knowledge. In the experiments, we fix the datasets size to 2G. Figure 4 shows the time consumption of generating frequent patterns by PACFP algorithms with the minimum support threshold of 0.1, 0.2, 0.3, 0.4, and 0.5 %.

Figure 4 shows that the execution time of the algorithm becomes shorter when the constraint condition is stricter under same minimum support threshold. This is because the number of records will greatly decrease when constraint condition is strengthened, and the storage space for building CFP-Tree will be reduced. So the running time of the algorithm will become shorter. At the same time, with the increase of the minimal support threshold, the execution time of the algorithm will reduce. It is because the amounts of qualified frequent patterns have decreased and the time of traversing CFP-Tree has reduced.

6.2 Scalability

Scalability is to verify the adaptability for different datasets size. In this experiment, we set the computing node to 4 and the minimum support threshold to 0.1, 0.2, 0.3, and 0.4 %, respectively. Figure 5 shows the time consumption of generating frequent patterns by the PACFP algorithm when the datasets vary from 1G to 5G by 1G, respectively.

As can be seen from Fig. 5, when the size of datasets is fixed, the execution time of algorithm becomes shorter under the stricter condition, which results from the scale of CFP-Tree becoming smaller and memory declining when the condition is stricter. Therefore, the mining efficiency improved dramatically. At the same time, with the increase of the number of records, the time of the first scan datasets and

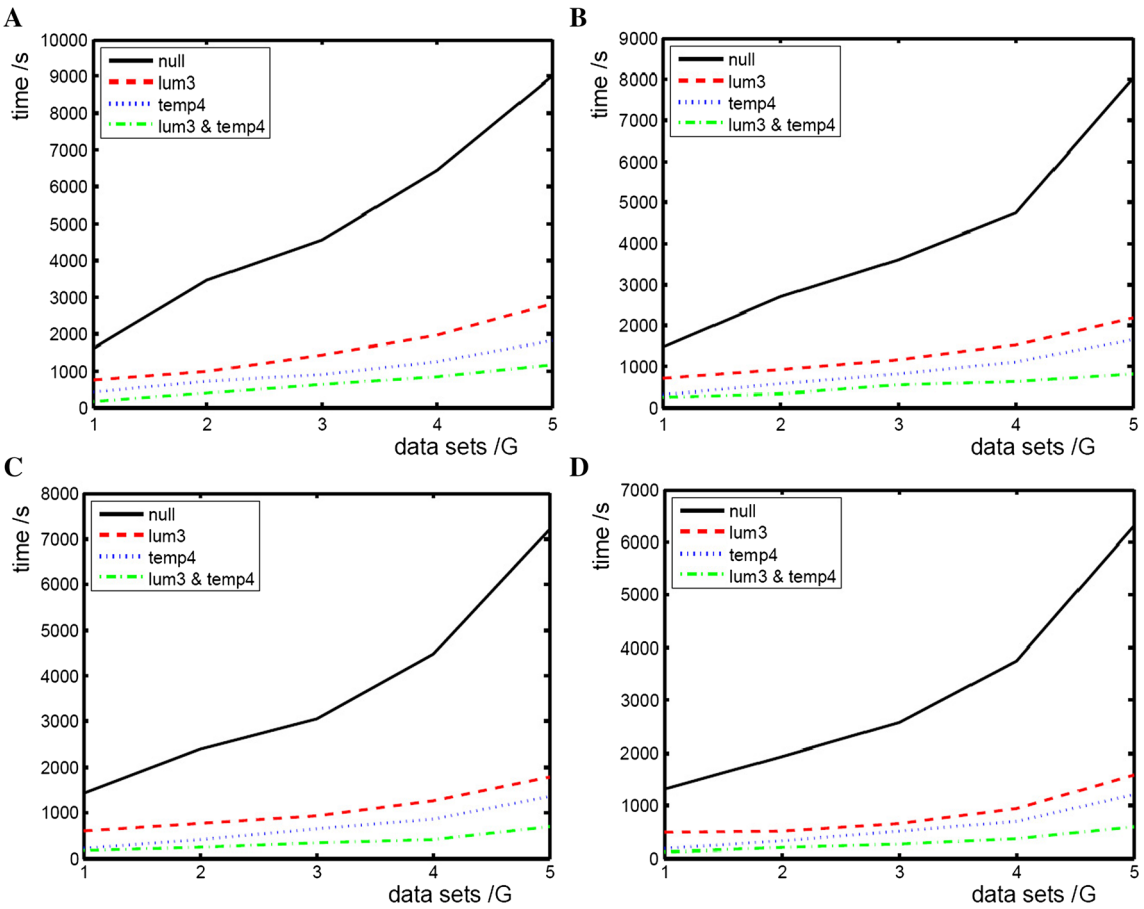

Fig. 5 The impact of datasets on the mining efficiency (node = 4) (The 4 subfigures are **a** support = 0.1 %; **b** support = 0.2 %; **c** support = 0.3 %; and **d** support = 0.4 % in sequence)

the amounts of qualified frequent patterns are significantly increased, which lead to the increase of the execution time of the PACFP algorithm.

6.3 Extensibility

Extensibility is used to verify the parallel algorithm's ability to adapt multiple computing nodes. In this experiment, we set the minimum support threshold to 0.1, 0.2, 0.3, and 0.4 %, respectively, and the different constraints are applied. Speedup is an effective measurement for extensibility. Figure 6 shows the speedups of the PACFP algorithm with a different number of nodes. To directly observe the relationship between the speedup of parallel algorithms and computing nodes, histograms were used to depict the experimental results.

As can be seen from Fig. 6, different constraint conditions have certain impacts on the speedup ratio of the parallel algorithm when the datasets and minimum support are fixed. With the change of constraint conditions, the scale of CFP-Tree, communication overhead, and the storage overhead will be changed. However, the speedup is to measure the relationship of running time between a single computing node and multiple nodes under the same conditions. So the change of the constraint conditions has no identified relations with the speedup. That is, though the nodes increase, the speedup ratios will increase slowly, which could be explained by the reason that the communication overhead will increase with the increase of nodes, and the speedup ratio will be decreased.

6.4 Cost analysis of judging constraints

CFP-Tree reduces the number of records in datasets redistributing, which effectively decreases communication cost, minimizes the records and the memory for constructing CFP-Tree. Thereby, the mining time will be saved. However, it is worth mentioning that the process of reduction is also time consuming. Figure 7 depicts the time costs on the judgment of constraint. Four different constraints are selected for comparison.

From Fig. 7, the time cost for judging constraint conditions is very limited. At the same time, datasets reduction based on constraint conditions is executed locally on each computing node. The size of datasets distributed in the first time remains

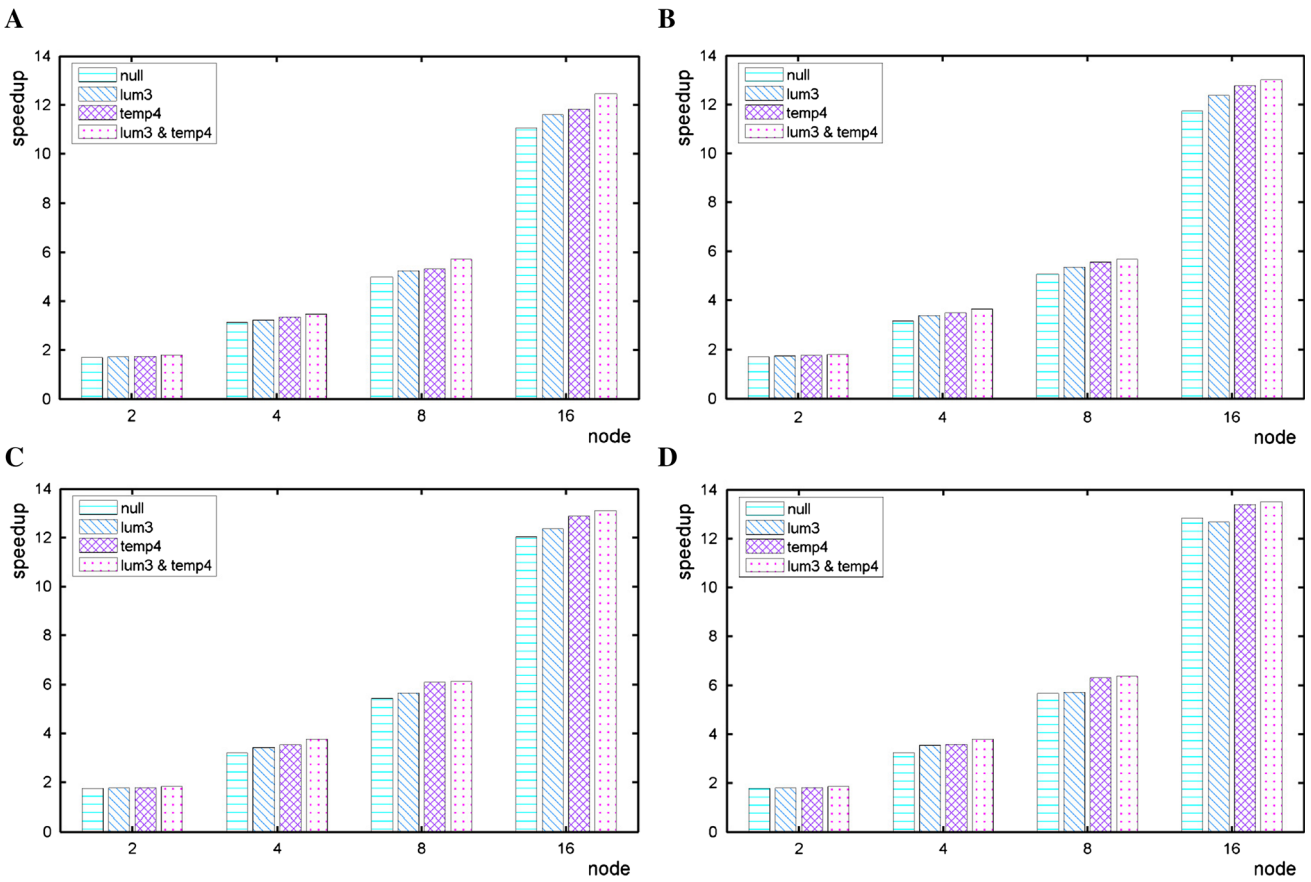

Fig. 6 The impact of nodes on the mining efficiency (datasets = 2G) (The 4 subfigures are **a** support = 0.1 %; **b** support = 0.2 %; **c** support = 0.3 %; and **d** support = 0.4 % in sequence)

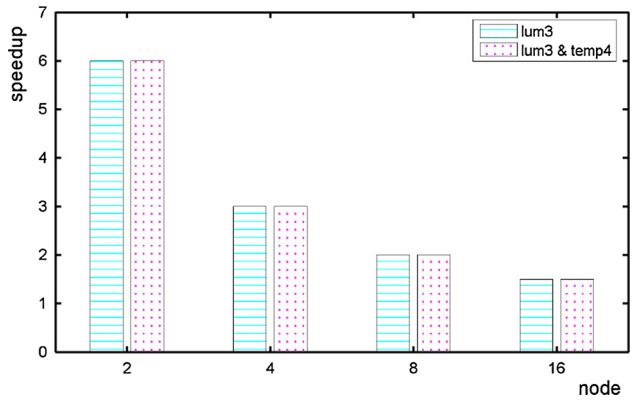

Fig. 7 The impact of nodes on the execution time of constraint judgment (datasets = 2G, minimum support threshold = 0. 1 %)

the same; therefore, the time for reduction linearly decreases with the increase of the amount of nodes.

6.5 Load balance

To get a well-balanced load and improve the ability of parallel processing and resource utilization in the cluster system, we introduce a load balancing technique from the perspective of data partitioning and distribution. In this experiment, we verify the impact of the load balancing strategy on execution time with different constraints. In Fig. 8, the algorithm with load balancing strategy depicted in 5.4 is marked as BPACFP and the algorithm without load balancing strategy is marked as PACFP. Minimum support threshold values were set to 0.1, 0.2, 0.3, 0.4, and 0.5 %, respectively.

As can be seen from Fig. 8, the load balancing strategy can effectively improve the efficiency of PACFP algorithm. We apply the data migration strategy to the original method; each computing node could be distributed frequent 1-itemsets with the same size. However, the frequency of frequent items is different, and communication cost and storage cost for constructing CFP-Tree are different for frequent items with different frequency which will lead to the problem of load imbalance among computing nodes. In the implementation process of the PACFP algorithm, a dynamic load balancing strategy is applied to construct the balanced CFP-Tree with the same size on each node according to the support count rather than the number of frequent items for achieving effective load balance.

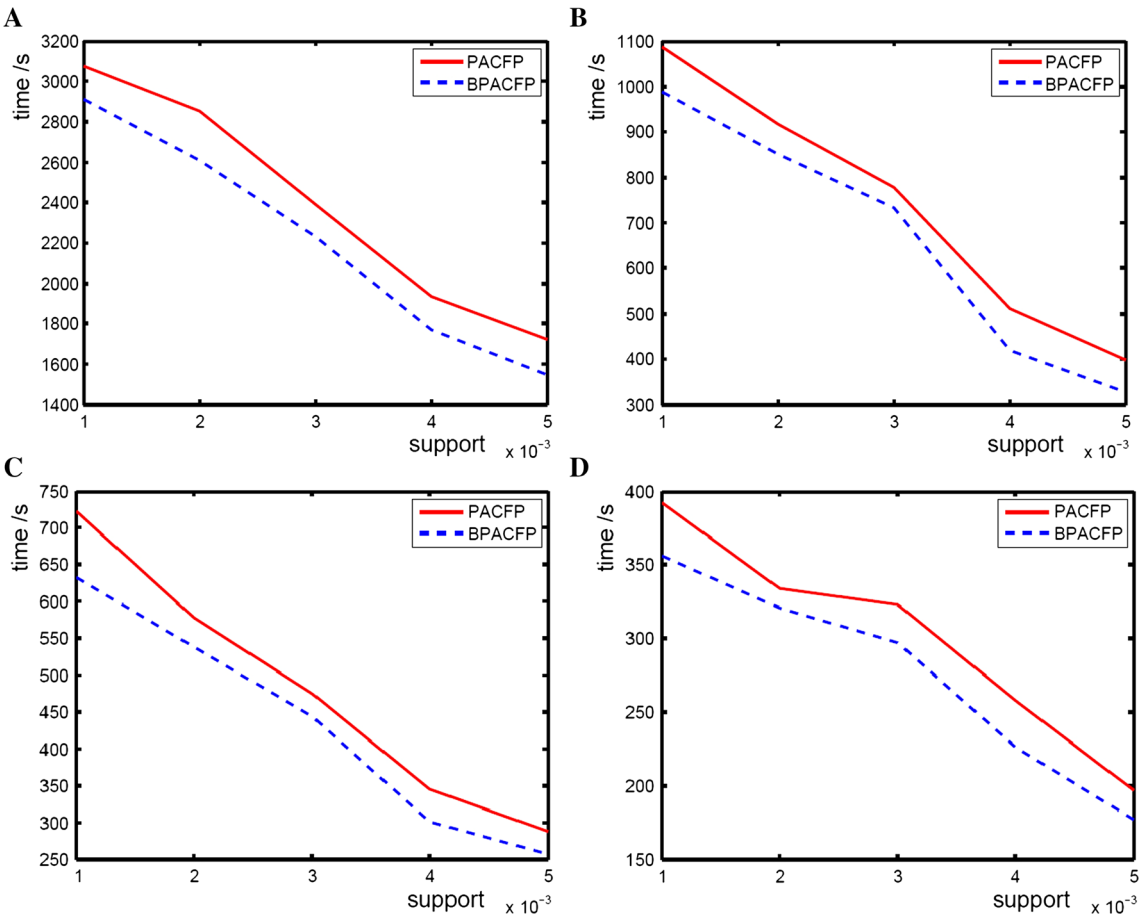

Fig. 8 The impact of load balancing strategy on the mining efficiency (datasets = 2G, nodes = 4) (The 4 subfigures are **a** null; **b** lum 3; **c** temp 4; and **d** lum 3 ∪ temp 4 in sequence)

7 Conclusion

Constrained frequent patterns, which have characteristics of stronger pertinence, higher practicability and mining efficiency, etc., refer to a kind of frequent pattern generated by constrained conditions given by users, and have been used in the celestial spectrum analysis. In this paper, we proposed a parallel mining algorithm of constrained frequent pattern under the MapReduce programming model on the Hadoop platform. Meanwhile, a load balancing strategy is presented for effective solving the load imbalance problems in our parallel implementation. Experimental results validate availability, scalability, and expandability of the algorithm using celestial spectrum datasets, so that an effective way is provided for improving the efficiency of association rules mining from massive high-dimensional datasets. In the future, our parallel mining algorithm will be used to discovering known and special celestial laws from the massive high-dimensional celestial spectrum datasets.

Acknowledgments This work is partially supported by the National Natural Science Foundation of P. R. China (61272263) and the Graduate Scientific and Technological Innovation Projects in TYUST (20134027). Xiao Qin's work is supported by the U.S. National Science Foundation under Grants CCF-0845257(CAREER), CNS-0917137(CSR), and CCF-0742187(CPA).

Compliance with ethical standards

Conflict of interest The authors declare that there is no conflict of interests regarding the publication of this paper.

References

Agrawal R, Imieliński T, Swami A (1993) Mining association rules between sets of items in large databases. ACM SIGMOD Record 22(2):207–216

Chen CC, Tseng CY, Chen MS (2013) Highly scalable sequential pattern mining based on mapreduce model on the cloud. In: 2013 IEEE international congress on big data (BigData Congress), pp 310–317

Chen MS, Han J, Yu PS (1996) Data mining: an overview from a database perspective. IEEE Trans Knowl Data Eng 8(6):866–883

Chen K, Zhang L, Li S, Ke W (2011) Research on association rules parallel algorithm based on fp-growth. In: Information computing and applications, pp 249–256

Dean J, Ghemawat S (2008) MapReduce: simplified data processing on large clusters. Commun ACM 51(1):107–113

Gao Y, Zhu S (2010) Improvement and realization of association rules mining algorithm based on FP-tree. In: 2010 2nd International conference on information science and engineering (ICISE), pp 1264–1266

Han J, Pei J, Yin Y (2000) Mining frequent patterns without candidate generation. ACM SIGMOD Record 29(2):1–12

Han J, Pei J, Yin Y, Mao R (2004) Mining frequent patterns without candidate generation: a frequent-pattern tree approach. Data Mining Knowl Discov 8(1):53–87

Han J, Kamber M (2006) Data mining. Concepts and techniques. Southeast Asia Edition

Hong S, Huaxuan Z, Shiping C, Chunyan H (2013) The study of improved FP-growth algorithm in MapReduce. In: 1st International workshop on cloud computing and information security

Hui-ling P, Yun-xing S (2012) A new FP-tree-based algorithm MMFI for mining the maximal frequent itemsets. In: 2012 IEEE international conference on computer science and automation engineering (CSAE), vol 2, pp 61–65

Islam ABMR, Chung TS (2011) An improved frequent pattern tree based association rule mining technique. In: 2011 International conference on information science and applications (ICISA), pp 1–8

Javed A, Khokhar A (2004) Frequent pattern mining on message passing multiprocessor systems. Distrib Parallel Databases 16(3):321–334

Lam C (2010) Hadoop in action. Manning Publications Co

Liu Y, Jiang X, Chen H, Ma J, Zhang X (2009) Mapreduce-based pattern finding algorithm applied in motif detection for prescription compatibility network. In: Advanced parallel processing technologies, pp 341–355

Li H, Wang Y, Zhang D, Zhang, M, Chang EY (2008) Pfp: parallel fp-growth for query recommendation. In: Proceedings of the 2008 ACM conference on recommender systems, pp 107–114

Rong Z, Xia D, Zhang Z (2013) Complex statistical analysis of big data: implementation and application of apriori and FP-growth algorithm based on MapReduce. In: 2013 4th IEEE international conference on software engineering and service science (ICSESS), pp 968–972

Seki K, Jinno R, Uehara K (2013) Parallel distributed trajectory pattern mining using hierarchical grid with MapReduce. Int J Grid High Perform Comput 5(4):79–96

Tu F, He B (2011) A parallel algorithm for mining association rules based on FP-tree. In: Advances in computer science, environment, ecoinformatics, and education, pp 399–403

Wang HJ, Hu CA (2010) Mining maximal patterns based on improved FP-tree and array technique. In: 2010 Third international symposium on intelligent information technology and security informatics (IITSI), pp 567–571

White T (2012) Hadoop: the definitive guide. O'Reilly Media, Inc

Yang XY, Liu Z, Fu Y (2010) MapReduce as a programming model for association rules algorithm on Hadoop. In: 2010 3rd International conference on information sciences and interaction sciences (ICIS), pp 99–102

Zhang J, Zhao X, Zhang S, Yin S, Qin X (2013) Interrelation analysis of celestial spectra data using constrained frequent pattern trees. Knowl Based Syst 41:77–88

Zhou J, Yu KM (2008) Tidset-based parallel FP-tree algorithm for the frequent pattern mining problem on PC clusters. In: Advances in grid and pervasive computing, pp 18–28

Zhou L, Zhong Z, Chang J, Li J, Huang JZ, Feng S (2010) Balanced parallel fp-growth with mapreduce. In: 2010 IEEE youth conference on information computing and telecommunications (YC-ICT), pp 243–246