International Journal for Housing Science and Its Applications
Publish August 5, 2025. Volume 46, Issue 3 Pages 3675-3686

https://doi.org/10.70517/ijhsa463310

Optimization of frequent itemset mining based on FP-growth algorithm

Yingchuan Liu[1,*]
[1] Public Education Department, Tangshan Preschool Teachers College, Tangshan, Hebei, 063000, China
Corresponding authors: (e-mail: 18942699391@163.com).

Abstract Frequent itemset mining plays an important role in many important data mining tasks. However, with the rapid development of big data, the demand for valuable information in data is increasing. Aiming at the problems of inefficiency and load unevenness of traditional FP-Growth algorithm for mining in big data environment, an improved parallel FP-Growth algorithm is proposed in this paper. The load balancing strategy is chosen as the solution to the problem, the optimized computational volume model is constructed considering the shortcomings that the computational volume model cannot reflect the characteristics of the data itself, and the optimized parallel FP-Growth algorithm is implemented under the Spark computing framework. The load-balancing based PFP algorithm is optimized to a great extent in terms of the energy consumption of the algorithm operation, and the energy consumption is reduced by up to 63.73% compared to the PFP algorithm. Excellent runtime distribution is obtained for a large number of tasks, and the runtime share of tasks is in a balanced distribution state. It illustrates the performance advantage of the algorithm in this paper, which can be effectively adapted to the frequent itemset mining of big data.

Index Terms Parallel FP-Growth algorithm, Load balancing strategy, Computation volume model, Spark computing framework

I. Introduction

In the digital era, the explosive growth of data volume has brought unprecedented challenges to the field of data mining [1], [2]. Frequent itemset mining, as one of the key techniques in data mining, aims to identify frequently occurring patterns or itemsets from a large amount of data, which are crucial for tasks such as understanding the intrinsic structure of data, supporting decision making, and predictive analytics [3]-[5]. However, with the increasing data size, how to mine these frequent itemsets efficiently and accurately has become an urgent problem [6]. Traditional frequent itemset mining algorithms, such as the Apriori algorithm, although theoretically groundbreaking, are computationally inefficient when dealing with large-scale datasets due to the need to generate a large number of candidate itemsets, which results in low computational efficiency and is very sensitive to parameter settings [7]. In addition, these algorithms are often difficult to capture the deep structure and complex patterns of data when dealing with high-dimensional data, limiting the accuracy and depth of mining results [8].

To address these challenges, many scholars at home and abroad have conducted in-depth research on frequent itemset mining algorithms.Karim et al [9] proposed the MFPAS algorithm, which employs a memory-efficient and faster method for constructing an efficient tree structure known as an ASP-Tree, with a smaller memory footprint requirement, reducing the space occupied for creating a conditional FP-tree. Mahdi, M. A et al [10] proposed FR-Tree algorithm which uses a new data structure F-Tree to compactly store and retrieve a large number of itemsets as a way to save main memory usage. Xun, Y et al [11] constructed a parallel Frequent Item Set Mining (FiDoop) algorithm based on MapReduce, which incorporates a frequent item supermetric tree to achieve compressed storage, avoiding the construction of conditional pattern libraries, and improving the performance of mining high-dimensional data. Han, X et al [12] proposed a pre-computation based frequent itemset mining (PFIM) algorithm which divides the transaction table into a large old table and a smaller new table and utilizes the quasi-frequent itemsets on the old table to quickly return the desired frequent itemsets, thus efficiently calculating the frequent itemsets on massive data. Fang, G et al [13] introduced the concept of minimum frequent closed particles and proposed an efficient frequent closed itemset mining algorithm, which effectively reduced the I/O and computational cost of the mining process. Although these algorithms optimize frequent itemset mining in time or space to some extent, they still have not fundamentally solved the problem of traditional FP-tree temporal and spatial inefficiency, and the optimization of the creation of conditional FP-tree process is still one of the important issues.

While the problem of creating conditional FP-trees with low spatial and temporal efficiency is an important issue to be considered, the performance of parallelization between nodes is also very significant in terms of its impact on the overall performance of the algorithm. To solve this problem, Alghyaline, S et al [14] proposed a new algorithm (FP-growth+) which improves on FP-growth and avoids the problem of creating conditional FP-trees. Elgaml, E. M et al [15] proposed an efficient multi-support frequent pattern growth algorithm that uses an infrequent child node pruning step so that the size of the spanning tree is smaller or equivalent to the size of the tree constructed by the FP-Growth method as a way to improve the performance of parallelization among nodes. Wang, C et al [16] proposed a parallel conditional frequent pattern algorithm that divides a large transactional database into multiple independent sub-databases, cyclically mines each sub-transactional database, and constructs the FP-tree in parallel, to achieve the goal of reducing the communication overhead between nodes and improving the performance of parallelization between nodes. Su, S et al [17] proposed a differential private frequent itemset mining (FIM) algorithm (PFP-growth) which consists of a preprocessing phase and a mining phase, where the preprocessing phase transforms the database using an intelligent segmentation method, and the mining phase employs runtime estimation and dynamic approximation, which offsets the problem of information loss and guaranteed privacy during the mining process. Thurachon, W and Kreesuradej, W [18] designed efficient association rule mining method which uses Fast Incremental Update Frequent Pattern Growth Algorithm (FIUFP-Growth), a new Incremental Conditional Pattern Tree (ICP-tree), and Compact Subtrees, which reduces the number of scans of the original database, and improves the execution speed over traditional algorithms by 46% or more. Although the above optimization algorithms improve the problem of redundant search in the mining process, they do not fundamentally solve the redundant search of the algorithms. Therefore, how to avoid redundant search and improve the efficiency of frequent itemset mining is still a problem that needs to be solved urgently.

Besides, how to set the support threshold dynamically is also a problem to be considered. To solve this problem, Chen, Y. C et al [19] proposed an improved FP-Growth based algorithm, which each item sets its own minimum support, and at the same time adopts an automatic multi-item support adjustment method based on the central limit theorem to adjust the support threshold. Bagui, S et al [20] proposed a heuristic based load balancing policy (HBPFP) in order to improve the performance and efficiency of parallel FP-growth (PFP) algorithm for frequent itemset discovery in big data. Leroy, V et al [21] proposed a new frequent itemset mining (TopPI) algorithm which efficiently mines the top-k most frequent closed itemsets of each item in the dataset, even if they are of very low frequency, in order to better capture long-tail features of the data. Qu, J. F et al [22] proposed an efficient frequent itemset mining (DPT) algorithm using only one dynamic prefix tree, which is more efficient than previous algorithms that require the construction of multiple prefix trees. Hong, T. P et al [23] proposed a new algorithm for mining fuzzy frequent term sets from transactions with quantitative values called Multiple Fuzzy FP-tree (MFFP-tree) algorithm, which is an improvement on the previous Fuzzy Frequent Pattern Tree (FP-tree) algorithm. Sun, J et al [24] proposed an incremental frequent itemset mining algorithm (FCFPIM) that uses a fully compressed frequent pattern tree (FCFP-Tree) data structure to maintain complete information about frequent and infrequent items so that the tree can be updated efficiently when new data arrives. In summary, although the above improved algorithms have made some progress in the process of frequent itemset mining, the issues of how to effectively improve the spatio-temporal efficiency of creating conditional FP-tree, how to reduce the communication overhead between nodes, how to solve the redundant search, and how to dynamically set the threshold of support are still urgent problems to be solved at present.

In order to optimize the load balancing performance of the parallel FP-Growth algorithm, this paper introduces a load balancing grouping strategy in the parallel FP-Growth algorithm. Then a prefix length-based computational amount model is introduced in this strategy, through which the computational amount generated by the items in the FP tree is reflected to reduce the error existing between the estimated results and the real results, so as to improve the ability and efficiency of the algorithm to process the data in big data. In order to adapt the optimized load balancing strategy and the parallel FP-Growth algorithm, the parallel FP-Growth algorithm is improved at three levels, namely, transaction ordering, prefix length grouping computation and corresponding condition division. Comparative tests are applied to verify the performance and load balancing effect of the improved algorithm.

II. Association Rule Mining Theory and FP-Growth Algorithm

II. A. Basic concepts of association rules

The purpose of association rule-based data mining is to discover the hidden associations between data in a large number of databases, and the associations discovered by this method can be expressed by association rules [25] or frequent itemsets. An expression such as $X \rightarrow Y$ is an association rule expression, in which X, Y has no intersection, i.e., they are not related to each other. And the level of association rule is expressed by support and confidence. Both are calculated as follows:

$$S = \frac{\sigma(X \cup Y)}{N} \tag{1}$$

$$C = \frac{\sigma(X \cup Y)}{\sigma(X)} \tag{2}$$

The support of the itemset X is denoted by $\sigma(X)$, which is the total amount of data that contains the itemset X in all the data, and is expressed by the formula:

$$\sigma(X) = |\{t_i \mid X \subseteq t_i, t_i \subseteq T\}| \tag{3}$$

The evaluation of association rules uses two variables, support and confidence, the main reason is that support for association rules, which represents the number of times an association rule is met in the item set, a large value of support is a prerequisite for the realization of an association rule, an association rule that meets the actual data a small number of times, i.e., a small value of support can not be counted as a valid rule, and there is no need to analyze it. Using support as a criterion reduces the number of unnecessary rules. In addition, the confidence level represents the theoretical basis for the prediction of what has not happened according to the rule, which has a high confidence level.

Data mining based on association rules can be simply described as follows: in a given dataset T, find the rule that occurs when $s \geq \min \sup$ (minsup is the support threshold) and $c \geq \min conf$ (minconf is the confidence threshold). What emerges according to the above principles is the only association rule that makes practical sense. The key to accomplishing this task is to compute s and c for all rules, based on which the eligible combinations can be found. However, this method is the most primitive and the most invested method, which is too difficult to realize. That's because for a dataset with a small number of rules, the number of various rules extracted from it is exponential, and for huge databases, the rules are conceivable.

II. B.Mining steps for association rules

A lot of association rule mining process is divided into two steps and there is a time sequence between the two parts of the work, first generating the frequent data sets that occur and then discovering the rules on their basis. Neither part of the work can be omitted, the discovery of rules is based on the frequent itemsets discovered in the first step, otherwise the rules cannot be generated. Usually, a term set containing k data items, there will be 2^{k-1} a frequent subset that is not the null value, in practice k value is very large, so the number of frequent term sets generated is very large. The most primitive way to find the frequent itemsets is to calculate the support of each dataset separately, so there is an urgent need to check each matter and observe whether it includes the candidate set, if yes, then the support for that candidate set is increased by one, and if not then the support remains unchanged. The input of this approach is high due to the fact that it has to go through O(NMw) comparisons and incomplete rules are generated.

On the basis of obtaining frequent itemsets, the next step of data mining, i.e., discovering rules, can be carried out. There is no need to consider the case where the antecedent and the consequent are empty because they have no use, for each frequent itemset object that has k, it can generate 2^{k-2} association rules. Association rules can be generated by first splitting Y into two subsets X and $Y-X$, so that it satisfies $X \rightarrow Y-X$ and so that it is greater than or equal to the minimum confidence value, as this condition is set up must be able to comply with the support threshold, that is because of its based on the frequent itemset [26], so it only needs to comply with the conditions of the confidence threshold can be.

II. C.Association Rule Mining Classification

The analysis of the shopping basket case is only a classic case study in association rule mining, there are many other kinds of data mining related to association rules in practical applications, which are often categorized into three main categories.

(1) According to the type of values handled by association rules

If there is an association rule that aims to discover the relationship between data items, then this rule is called Boolean, such as {milk, eggs}→{bread} in the case study is a typical example of the above rule. The rule only describes the relationship between the data and has nothing to do with the amount of data.

(2) According to the number of data dimensions in the association rule

If there is a correlation rule of the antecedent and the antecedent is the same dimension of the data, then it is called a single-latitude correlation rule, like {milk}→{bread}, which has only one dimension of the goods purchased by the consumer, it belongs to the single-dimension correlation.

(3) According to the abstraction level of the description content

If there is an association rule of the rule before and after the piece of data is the same level, it is called a single-level association rule, like {milk} → {bread}, the rule of the data items milk and bread is the same level of data, so it is a single-level association rule.

II. D.FP-Growth Algorithm

With the support of FP-tree [27] FP-Growth algorithm can directly determine whether the itemset is frequent or not, so that the candidate itemset generation link is meaningless, and then most of the process simplification purposes.

Principle of the algorithm: the first part of the algorithm is the construction of FP-tree tree, FP-tree represents the compression of data.

First of all, each transaction is read one by one, and then the transaction is mapped on a certain path on the FP-tree, even if the transaction is different, its antecedent may be uniform, then the FP-tree specific path on the probability of overlap is relatively large, the larger the probability of overlap, proving that the results of data compression is more significant, representing the construction of the FP-tree is reasonable. The optimal state is that all transactions are identical, or at least the prefixes are completely uniform, in which case there is only one path mapped on the FP-tree. The worst case is that all transactions are not uniform at all, in which the size of transaction dataset and FP-tree are completely uniform, and the function of data compression is not reflected.

After the FP-tree is constructed, the FP-Growth algorithm is used to analyze the FP-tree in bottom-up order to generate the frequent itemsets based on the FP-tree. Compared to the Apriori algorithm, FP-Growth algorithm has some differences, Apriori algorithm is in the frequent 1-item set, frequent 2-item set ⋯ generated frequent itemset generation, FP-Growth algorithm is the end of the term as a foothold for in-depth excavation of the frequent itemset, based on partitioning strategy for the refinement of the frequent itemset mining of this major problem, will be further divided into similar based on a certain item, and will be used to generate the FP-tree. It is further divided into branching problems similar to mining all frequent itemsets based on the ending of a certain item, which is finally handled properly.

III. Load balancing-based optimization of parallel FP-Growth algorithm

In this paper, a typical association rule algorithm, i.e., FP-Growth algorithm, is studied in depth. However, the traditional serial FP-Growth algorithm is still unable to solve the association rule mining problem under big data, and it is always limited by hardware resources under a single node, and suffers from common problems such as memory bottleneck and insufficient computational power. Therefore, this paper shifts the focus of work to the research area of parallel FP-Growth algorithm [28].

III. A. Parallel FP-Growth Algorithm

III. A. 1) Parallelization of ideas

Parallel algorithms can be divided into two categories based on task decomposition and data decomposition, and parallel FP-Growth algorithm is realized based on the latter. Because, the Hadoop, Spark platform divides the massive data into blocks and stores them on each node in the cluster, eliminating the task of data decomposition for the parallel FP-Growth algorithm. At this point, the performance of the whole algorithm is significantly improved by each node directly mining its own stored dataset, and then getting the final result from the local results.

III. A. 2) Design Options for the Parallel FP-Growth Algorithm

The traditional FP-Growth algorithm consists of two main processes:(1) Counting frequent l-item sets and creating FP trees. (2) Construct conditional FP trees based on FP trees and recursively mine frequent l-term sets. The implementation of the parallel FP-Growth algorithm also consists of two processes: (1) Counting frequent 1-item sets in parallel. (2) Mining local conditional FP trees in parallel.

The decomposition strategy used in this paper is identified below. The division strategy is the most basic implementation, which needs to construct the local FP tree first, and then find the conditional pattern base Vertical projection needs to carry out the projection operation on the set of conditional pattern bases sequentially, which can only be in a serial way. While horizontal projection can directly perform projection operations on conditional pattern bases at multiple nodes at the same time, so this strategy is used in this paper to construct the local conditional FP tree.

III. A. 3) The PFP algorithm and problems with it

Although the PFP algorithm only mines the first K frequent patterns with the highest number of supports, it makes full use of the parallelization scheme and is a typical parallel FP-Growth algorithm, but it also has two major problems:

(1)The intermediate results generated by MapReduce tasks need to be saved to disk, and the next task then reads from disk, and the large number of VO reads and writes reduces the computational efficiency.

(2) The PFP algorithm adopts a relatively simple grouping strategy, and does not consider that different frequent items may generate different computational volumes, which in turn leads to uneven loads between groups, and large differences in the execution time of different nodes, affecting the execution efficiency of the parallel algorithm.

For Problem 1, the common solution is to use the Spark computing framework based on memory computing. And the solution to Problem 2 requires the introduction of a load balancing strategy, which does not only consider the similarity in the number of items, but also combines the computational volume of frequent items to group them so as to ensure that the running time of different computing nodes is similar.

III. B. Load Balancing Strategy Optimization

To address the problem of uneven grouping, some scholars propose a load balancing strategy, i.e., first build a computational volume model to estimate the computational maximum smallness of each frequent item in the construction and mining process of conditional FP tree, and then divide the frequent items into groups according to the greedy strategy.

III. B. 1) Load Balancing Strategy

The core of the load balancing strategy [29] is to evaluate the size of the computation for recursive mining of the conditional FP tree of each frequent item, and then use the grouping strategy to classify all frequent items into different groups, and use the decomposition strategy to decompose all the things into different groups and then each computational node is responsible for completing the corresponding grouping task, mining the conditional FP tree of the frequent items within the group, and finally ensure that the load of each computational node is The load of each computing node is balanced so that the running time of each node is similar, thus improving the execution efficiency of the parallel algorithm.

Firstly, from the theoretical point of view, the load balancing strategy can effectively improve the performance of parallel algorithms. Assuming that the running time of a step in the parallel process is T, and the task of this step is completed by n nodes, and using t_i to denote the execution time required by the i node to complete the task, then:

$$T = \max\{t_1, t_2, \cdots, t_i, \cdots, t_n\} \tag{4}$$

$$\overline{t} = \left(\sum_{i=1}^{s} t_i\right) / s \tag{5}$$

$$T_{\min} = \overline{t} \tag{6}$$

In the order from bottom to top in the F_List, the frequent items are calculated from large to small.

$$L_i = \log(P_i) \tag{7}$$

Then, the balanced grouping strategy is analyzed. The commonly used grouping strategy is realized based on the greedy strategy, each time the frequent items to be grouped are divided into the group with the smallest sum of computation, and at the same time the computation of the item is accumulated to the sum of computation of the group in which it is located, until all the items are finished grouping.

Finally, according to the grouping result GList, the decomposition strategy is executed to decompose the conditional pattern bases of different frequent items from the transaction and project them into the corresponding sets, and then the conditional FP trees of different frequent items are mined to find the local frequent item sets.

III. B. 2) Computational volume model optimization

The computational volume model is only related to the position of the items in the chain header table and does not reflect the characteristics of the data itself. This problem is mainly due to the fact that the algorithm only considers the effect of the depth of the P-tree on recursive mining and ignores the effect of the width of the FP-tree on recursive mining. If this load-balancing strategy is used to group the sample dataset, the grouping results obtained, in which mining the frequent itemsets with r as the pattern suffix, should generate a larger computational amount than that of s. The resulting grouping still has a certain degree of load inequality, and for this reason, this paper proposes a computational amount model based on the length of the prefix, while reflecting the computational amount of the items in the FP-tree that is generated by the length of the path in conjunction with multipaths. Computation Volume.

In order to apply the optimized load balancing strategy to the PFP algorithm, this paper makes several improvements to the PFP algorithm process to maintain the same number of scans of the transaction dataset and ensure a certain degree of parallelization efficiency. The specific improvements are as follows:

(1) Perform flatMap operation on the sorted transaction, then generate a collection of mutually independent conditional pattern bases by groupByKey(), which is used to construct a conditional FP tree for each item, calculate the prefix length, and save the prefix length and the conditional FP tree into the key-value pairs corresponding to the items.

(2) Grouping calculation of prefix lengths is performed item by item according to the bottom-up order in the chain header table, and each time the prefix lengths of the new items are accumulated into the group with the smallest total length, and at the same time the item is given the corresponding group number gid until the end of the grouping:

(3) Based on the <key: group number, value, the root node of the conditional FP tree corresponding to each item divided into the group> stored in the G_List of the hash structure, the FP_growth() function is called to mine the set of frequent items with each item as the pattern suffix.

The PFP algorithm is implemented under Spark based on an optimized load balancing strategy and is notated as LBPFP algorithm.

IV. Algorithm performance experiment and analysis

IV. A. Experimental environment and data set

For the algorithm proposed in this paper, a series of experiments are designed to verify its performance. The experimental hardware configuration is a distributed cluster with a master-slave structure, and all the nodes are configured with Intel(R) Core(TM) i7-8750H, 2.20GHz 8-core CPU, 16GB RAM, and 512GB storage hard disk, and they are all in the same LAN. The software configuration is uniformly installed with Hadoop 2.7.7, Scala 2.11.0, Spark 2.0.0, and Python 3.7.

The performance of the algorithms in this paper needs to be verified on datasets of different sizes, and in the comparison of algorithm performance, different data volumes and dimensions of the data are required in order to make the presentation of the experimental results in terms of the variation of the differences in the algorithms more obvious. Therefore, in this paper, experiments are conducted using three real datasets, namely Wikipedia, LiveJournal and webdocs. Wikipedia is a free encyclopedia written by volunteers from various countries. LiveJournal is a free online blogging community. webdocs is a web document data.

IV. B. Comparative Performance Analysis of FP-Growth Algorithm

In order to analyze the performance of FP-Growth algorithm for mining frequent itemsets in a big data environment, comparative analyses of runtime, memory usage, and speedup ratio are performed on the datasets Wikipedia, LiveJournal, and webdocs, respectively, compared with FR-Tree algorithm, MSFP-growth algorithm, and FP-ME algorithm.

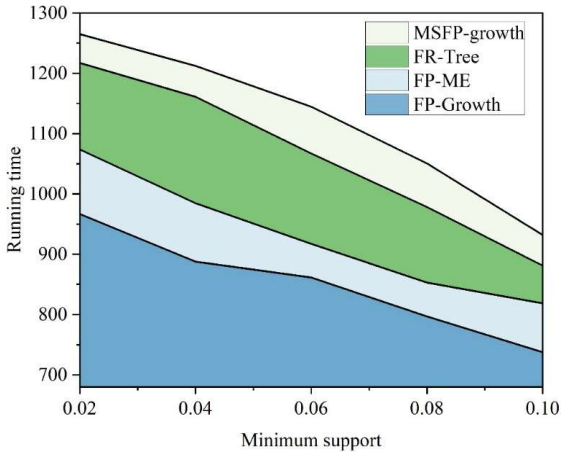

Figure 1: Runtime on Wikipedia

IV. B. 1) Comparative analysis of running time

The number of nodes is set to 4. The FP-Growth algorithm is compared with FR-Tree algorithm, MSFP-growth algorithm and FP-ME algorithm on the datasets Wikipedia, LiveJournal and webdocs for the runtime comparison, and the results of the comparison experiments are shown in Fig. 1-Fig. 3, respectively.

The running time of each algorithm decreases gradually as the support threshold increases. The FP-Growth algorithm always has the smallest running time, and as the dataset increases, the more obvious the advantage of FP-Growth algorithm over MSFP-growth, FR-Tree and FP-ME algorithms in terms of running time. In Fig. 3, the

running time of FP-Growth algorithm is reduced by 56.15% compared to MSFP-growth algorithm when the minimum support is 0.02. FP-Growth algorithm reduces the FP-tree structure of each node, which improves the efficiency of mining frequent itemsets and makes the algorithm's running time reduced. Therefore, the running time of FP-Growth algorithm in processing these datasets is always minimized compared to other algorithms.

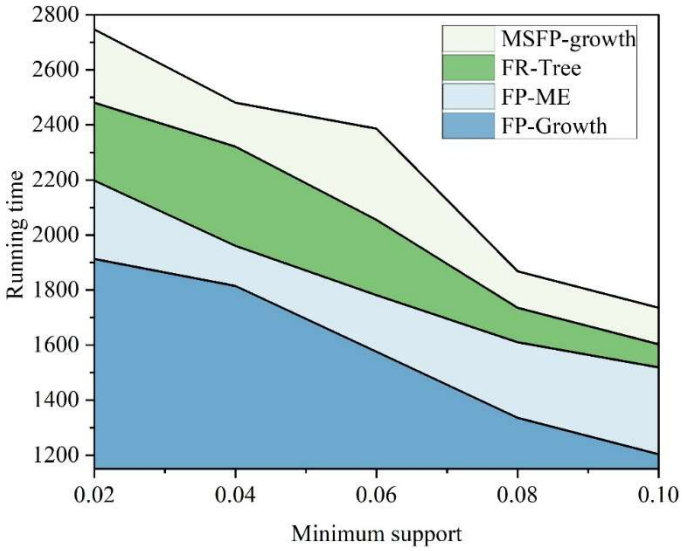

Figure 2: Runtime on LiveJournal

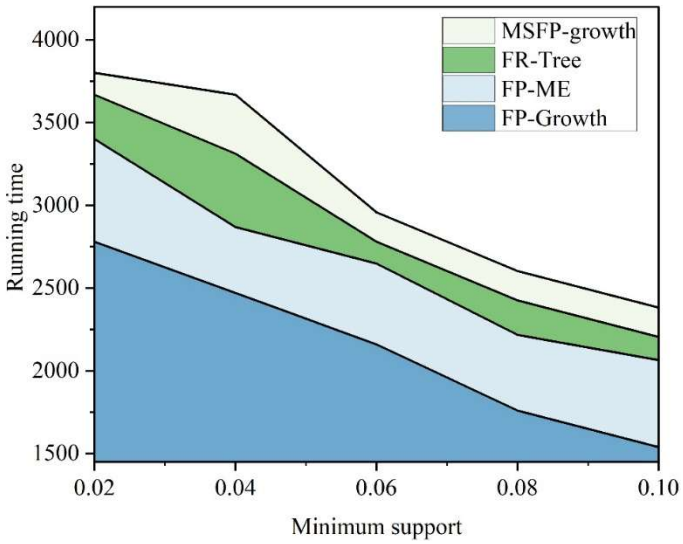

Figure 3: Runtime on webdocs

IV. B. 2) Comparative analysis of memory usage

Setting the number of nodes to 4, the FP-Growth algorithm is compared with the FR-Tree algorithm, the MSFP-growth algorithm and the FP-ME algorithm in terms of memory footprint on the datasets Wikipedia, LiveJournal, and webdocs, and the experimental structure is shown in Fig. 4-Fig. 6, respectively.

The memory footprint of each algorithm decreases gradually as the minimum support is supported. Among these algorithms, FP-Growth algorithm always has the lowest memory footprint. For example, when processing the dataset webdocs with a minimum support of 0.02, the FP-Growth algorithm reduces the memory footprint by 16.95%, 15.12%, and 13.27%, respectively, as compared to the MSFP-growth algorithm, the FR-Tree algorithm, and the FP-ME algorithm.The main reason for the superiority of the FP-Growth algorithm in terms of space complexity is that the FP -Growth algorithm adopts a pruning strategy that uniformly groups the frequent1 item sets and shrinks the FP-tree structure generated by each grouping, which makes the algorithm satisfy fewer frequent items under the same support threshold, and reduces the memory occupancy of frequent item storage from the source. As a result,

the memory usage of FP-Growth algorithm is always minimized compared to other algorithms when processing each dataset.

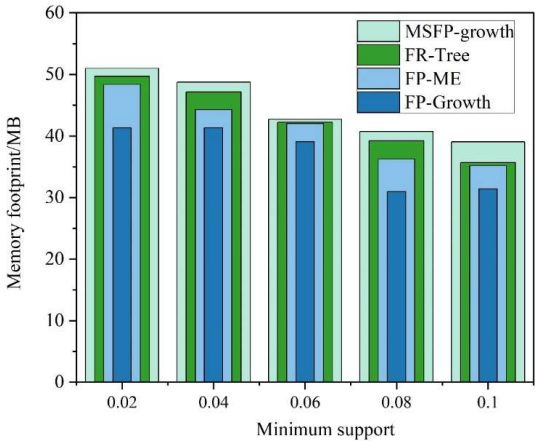

Figure 4: Wikipedia's memory ratio

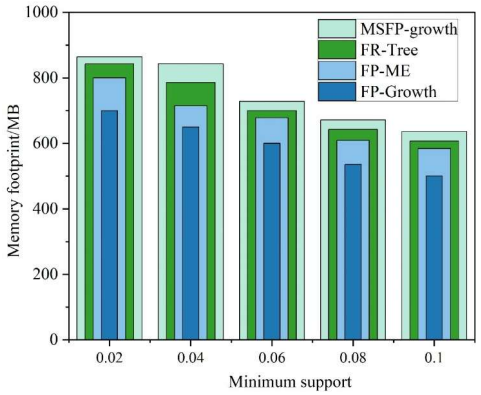

Figure 5: LiveJournal's memory ratio

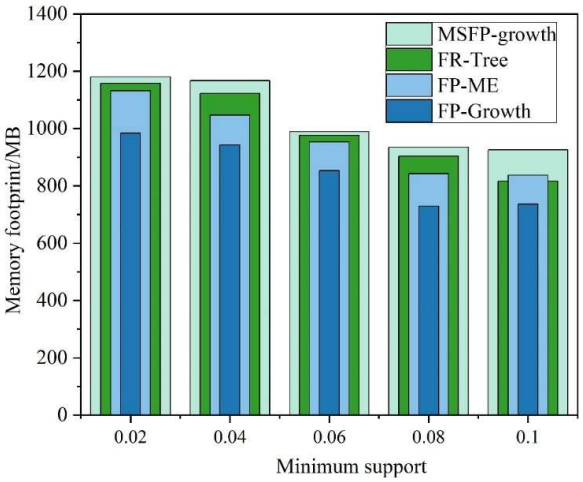

Figure 6: Webdocs's memory ratio

IV. B. 3) Comparative analysis of acceleration ratios

The acceleration ratio of FP-Growth algorithm is compared with FR-Tree algorithm, MSFP-growth algorithm and FP-ME algorithm on the datasets Wikipedia, LiveJournal, and webdocs, and the experimental results are shown in Fig. 7-Fig. 9.

The FP-Growth algorithm always has the highest speedup ratio for mining frequent items from different datasets. When dealing with the webdocs dataset, the acceleration ratio of FP-Growth algorithm reaches the highest of 2.67 when the number of nodes is 5, which is an increase of 0.67, 0.47, and 0.22, respectively, compared with MSFP-growth algorithm, FR-Tree algorithm, and FP-ME algorithm.The reasons for the better acceleration ratio of FP-Growth algorithm compared with the other algorithms are: firstly , when dealing with large datasets, as the number of nodes increases, the less frequent items are assigned to each node, resulting in the algorithm's running time for mining frequent itemsets in multiple nodes is lower than that of a single node, which improves the algorithm's speedup ratio. Secondly, the FP-Growth algorithm removes paths in the FP-tree that do not satisfy the supercluster pattern, which reduces the traversal time of the tree, thus improving the efficiency of the algorithm to mine frequent itemsets in parallel and increasing the speedup ratio. As a result, the FP-Growth algorithm consistently has the highest speedup ratio compared to the other algorithms when processing the datasets Wikipedia, LiveJournal, and webdocs.

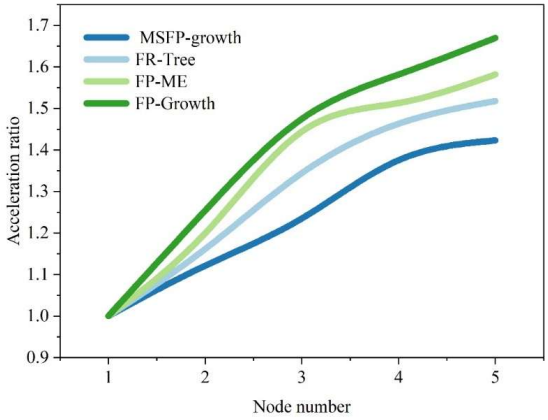

Figure 7: Wikipedia's acceleration ratio

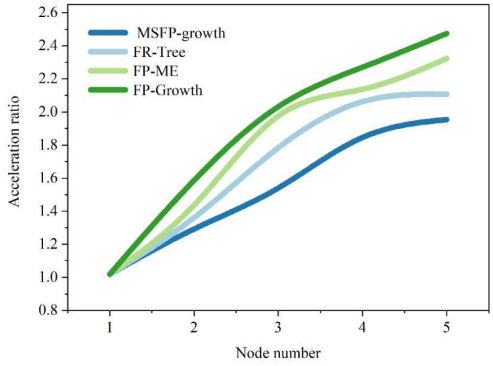

Figure 8: LiveJournal's acceleration ratio

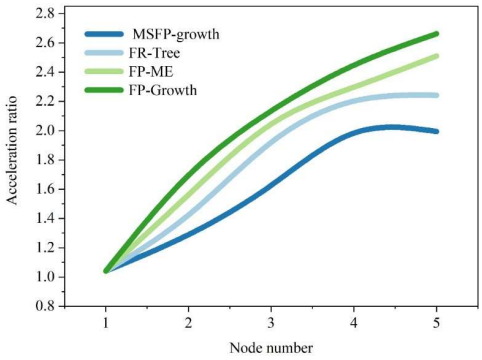

Figure 9: Webdocs's acceleration ratio

IV. C. Performance test of PFP algorithm based on load balancing

In this paper, in order to test the load balancing based PFP algorithms, benchmark algorithms need to be selected. In this paper the selection is done among several algorithms. Because load balancing based PFP is a parallel algorithm, in order to better test, the first selection criterion is to select the parallel algorithm, if the serial algorithm is selected, it can not reflect the advantages of load balancing based PFP algorithm. Among the parallel algorithms, one of the most classic algorithms is of course the PFP algorithm, which is used as one of the benchmark algorithms in this paper to compare with the load-balancing-based PFP algorithm.Prepost algorithm is a serial algorithm with excellent results in recent years, and in order to compare this paper's load-balancing-based PFP algorithm with Prepost, this paper also uses the MRPrePost algorithm as a benchmark algorithm. However, because the MRPrePost algorithm is parallelized on the MapReduce platform, the advantages of this algorithm can not be better reflected, so this paper also does an additional work, which is to parallelize Prepost on the Spark platform according to the parallelism of the MRPrePost, to improve the performance of the original MRPrepost algorithm, and to rule out the influence of the platform on the algorithm, in order to facilitate the follow-up of the algorithm, we also use the MRPrePost algorithm as a benchmark algorithm. In order to facilitate the description of the subsequent tests, this paper will still refer to the Prepost algorithm parallelized on the Spark platform as the MRPrepost algorithm.

IV. C. 1) Energy consumption tests

Since the memory consumption gap between the algorithms is too large, the logarithm of the memory consumption is taken with a base of 2 to plot the results of the memory overhead test and the results are shown in Table 1.

Firstly, the memory overhead of load balancing based PFP algorithm, PFP algorithm and MRPrepost algorithm under Wikipedia is tested for each support level and these three algorithms are tested under 5 support levels of 20%, 30%, 40%, 50% and 60% respectively.

Load balancing based PFP always outperforms PNP algorithm and MRPrepost algorithm in terms of memory consumption in case of Wikipedia data and PFP algorithm has the worst performance, while load balancing based PFP algorithm reduces the memory overhead by a maximum of 63.16% with respect to PFP algorithm and by a maximum of 52.82% with respect to MRPrepost algorithm.

Table 1: The algorithm's energy consumption test compares results

Data set	Support/%	PFP	MRPrepost	PFP based on load balancing
Wikipedia	20	450	356	211
	30	405	341	198
	40	344	323	165
	50	312	301	142
	60	285	158	105
LiveJournal	1	1250	1000	654
	5	1156	854	589
	10	1052	653	496
	15	894	511	356
	20	801	443	298
Webdocs	0.2	2250	1854	1400
	0.4	2113	1654	1256
	0.6	2001	1523	1023
	0.8	1911	1245	854
	1.0	1803	1356	654

Then the memory overhead of each support degree under LiveJournal is tested for load balancing based PFP algorithm, PFP algorithm and MRPrepost algorithm. The three algorithms are tested under five support levels of 1%, 5%, 10%, 15% and 20% respectively.

Load balancing based PFP always outperforms PNP algorithm and MRPrepost algorithm in terms of memory consumption in case of LiveJournal data and PFP algorithm has the worst performance, while load balancing based PFP algorithm reduces the memory overhead by a maximum of 62.80% with respect to PFP algorithm and by a maximum of 32.73% with respect to MRPrepost algorithm.

Then the memory overhead of load balancing based PFP algorithm, PFP algorithm and MRPrepost algorithm under webdocs is tested for each support level and these three algorithms are tested under 5 support levels, which are 0.2%, 0.4%, 0.6%, 0.8% and 1% respectively.

Load balancing based PFP consistently outperforms PNP algorithm and MRPrepost algorithm in terms of memory consumption in case of webdocs data and PFP algorithm has the worst performance while load balancing based PFP algorithm reduces the memory overhead by a maximum of 63.73% in comparison to the PFP algorithm and by a maximum of 51.77% in comparison to the MRPrepost algorithm. On webdocs dataset, load balancing based PFP is always better than MRPrepost algorithm in terms of memory consumption.

Based on the above 3 experiments on memory overhead it can be shown that load balancing based PFP algorithm always outperforms PFP algorithm and also always outperforms MRPrepost algorithm.

IV. C. 2) Load balancing tests

In order to compare the benefit of load balancing strategy in this paper, this paper also obtains the task time of load balancing based PFP algorithm and MRPrepost algorithm under the same dataset, the same support degree and the same number of subgroups to see whether the load balancing strategy has some effect. Randomly selecting the dataset and support degree, this paper takes Wikipedia dataset, support degree is 20%, as well as the number of subgroups is 30 groups, finally the running time distribution of MPRrepost and load balancing based PFP can be shown in Fig. 10.

The running time of the 50 tasks of the load balancing based PFP is more uniform, and the running time percentage of all 50 tasks is 2%, which indicates that the load balancing strategy plays a certain role in making each machine with similar loads and close running time to achieve load balancing and shorten the running time of the algorithm.

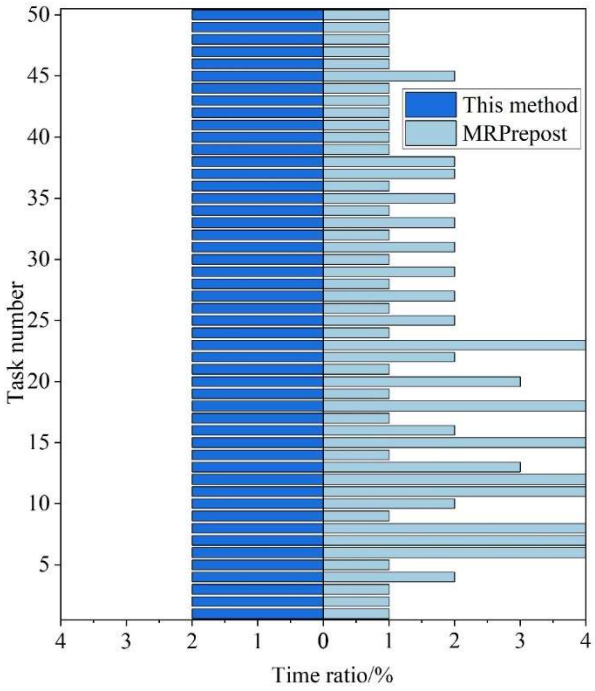

Figure 10: Running time ratio

V. Conclusion

In this paper, FP-Growth algorithm is proposed based on association rule theory, and the parallel FP-Growth algorithm is designed to address the problem of uneven load in its parallelization process.

(1) Comparative experiments are designed to verify the performance of FP-Growth algorithm on frequent itemset mining.FP-Growth algorithm constructs a tree structure by mining the generated frequent items, and compared with MSFP-growth algorithm, its running time is reduced by 56.15%, and the memory occupancy ratio are all reduced to different degrees. In the webdocs dataset, the speedup ratio of this paper's algorithm increases to a degree of 0.20 or more at a node number of 5, compared to comparative algorithms such as MSFP-growth. Better performance than FR-Tree algorithm, MSFP-growth algorithm and FP-ME algorithm is obtained on different three datasets.

(2) Load balancing based PFP consistently outperforms PFP and MRPrepost algorithms in terms of energy consumption for algorithm operation. Compared to PFP algorithm, the energy consumption of this paper's algorithm is minimized by 63.73%. And it is able to distribute the running time of 50 tasks in a balanced way, so that the

running time share of each task is 2%. It shows that the load balancing strategy of this paper's algorithm is effective and can balance the decision-making running time ratio so that the algorithm achieves load balancing. Overall, the load balancing-based PFP achieves excellent performance and is more suitable for big data frequent itemset mining than the comparison algorithm proposed in this paper.

References

[1] Wu, X., Zhu, X., Wu, G. Q., & Ding, W. (2013). Data mining with big data. IEEE transactions on knowledge and data engineering, 26(1), 97-107.

[2] Zhong, Y., Chen, L., Dan, C., & Rezaeipanah, A. (2022). A systematic survey of data mining and big data analysis in internet of things. The Journal of Supercomputing, 78(17), 18405-18453.

[3] Chee, C. H., Jaafar, J., Aziz, I. A., Hasan, M. H., & Yeoh, W. (2019). Algorithms for frequent itemset mining: a literature review. Artificial Intelligence Review, 52, 2603-2621.

[4] Apiletti, D., Baralis, E., Cerquitelli, T., Garza, P., Pulvirenti, F., & Venturini, L. (2017). Frequent itemsets mining for big data: a comparative analysis. Big data research, 9, 67-83.

[5] Rana, I., & Rathod, A. (2016). Frequent Item Set Mining In Data Mining: A survey. International Journal on Recent and Innovation Trends in Computing and Communication, 4(3), 124-126.

[6] Xie, H. (2021). Research and case analysis of apriori algorithm based on mining frequent item-sets. Open Journal of Social Sciences, 9(04), 458.

[7] Zeghina, A., Leborgne, A., Le Ber, F., & Vacavant, A. (2023). Multi-SPMiner: A Deep Learning Framework for Multi-Graph Frequent Pattern Mining with Application to spatiotemporal Graphs. Procedia Computer Science, 225, 1094-1103.

[8] Luna, J. M., Fournier-Viger, P., & Ventura, S. (2019). Frequent itemset mining: A 25 years review. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery, 9(6), e1329.

[9] Karim, M. R., Cochez, M., Beyan, O. D., Ahmed, C. F., & Decker, S. (2018). Mining maximal frequent patterns in transactional databases and dynamic data streams: a spark-based approach. Information Sciences, 432, 278-300.

[10] Mahdi, M. A., Hosny, K. M., & Elhenawy, I. (2022). FR-Tree: A novel rare association rule for big data problem. Expert Systems with Applications, 187, 115898.

[11] Xun, Y., Zhang, J., & Qin, X. (2015). Fidoop: Parallel mining of frequent itemsets using mapreduce. IEEE transactions on Systems, Man, and Cybernetics: systems, 46(3), 313-325.

[12] Han, X., Liu, X., Chen, J., Lai, G., Gao, H., & Li, J. (2019). Efficiently mining frequent itemsets on massive data. IEEE Access, 7, 31409-31421.

[13] Fang, G., Wu, Y., Li, M., & Chen, J. (2015). An efficient algorithm for mining frequent closed itemsets. Informatica, 39(1).

[14] Alghyaline, S., Hsieh, J. W., & Lai, J. Z. (2016). Efficiently mining frequent itemsets in transactional databases. Journal of Marine Science and Technology, 24(2), 12.

[15] Elgaml, E. M., Ibrahim, D. M., & Sallam, E. A. (2015). Improved FP-growth algorithm with multiple minimum supports using maximum constraints. World Acad Sci Eng Technol Int J Comput Electr Autom Control Inf Eng, 9(5), 1087-1094.

[16] Wang, C., Bian, W., Wang, R., Chen, H., Ye, Z., & Yan, L. (2020). Association rules mining in parallel conditional tree based on grid computing inspired partition algorithm. International Journal of Web and Grid Services, 16(3), 321-339.

[17] Su, S., Xu, S., Cheng, X., Li, Z., & Yang, F. (2015). Differentially private frequent itemset mining via transaction splitting. IEEE transactions on knowledge and data engineering, 27(7), 1875-1891.

[18] Thurachon, W., & Kreesuradej, W. (2021). Incremental association rule mining with a fast incremental updating frequent pattern growth algorithm. IEEE access, 9, 55726-55741.

[19] Chen, Y. C., Lin, G., Chan, Y. H., & Shih, M. J. (2014, November). Mining frequent patterns with multiple item support thresholds in tourism information databases. In International Conference on Technologies and Applications of Artificial Intelligence (pp. 89-98). Cham: Springer International Publishing.

[20] Bagui, S., Devulapalli, K., & Coffey, J. (2020). A heuristic approach for load balancing the FP-growth algorithm on MapReduce. Array, 7, 100035.

[21] Leroy, V., Kirchgessner, M., Termier, A., & Amer-Yahia, S. (2017). TopPI: an efficient algorithm for item-centric mining. Information Systems, 64, 104-118.

[22] Qu, J. F., Hang, B., Wu, Z., Wu, Z., Gu, Q., & Tang, B. (2020). Efficient mining of frequent itemsets using only one dynamic prefix tree. IEEE Access, 8, 183722-183735.

[23] Hong, T. P., Lin, C. W., & Lin, T. C. (2014). THE MFFP-TREE FUZZY MINING ALGORITHM TO DISCOVER COMPLETE LINGUISTIC FREQUENT ITEMSETS. Computational Intelligence, 30(1), 145-166.

[24] Sun, J., Xun, Y., Zhang, J., & Li, J. (2019). Incremental frequent itemsets mining with FCFP tree. IEEE Access, 7, 136511-136524.

[25] Lijuan He,Weimin Wang,Xiaojuan Wang,Deyin Zhang,Yukun Zhang,Yuan Zhao... & Xiaoxue Zhang. (2025). Association analysis of single nucleotide polymorphisms of LMOD3 gene and feed efficiency in sheep. Small Ruminant Research,247,107494-107494.

[26] Deng Fan,Wang Jiabin & Lv Sheng. (2024). Optimization of frequent item set mining parallelization algorithm based on spark platform. Discover Computing,27(1),38-38.

[27] Ghosh Moumita,Roy Anirban,Sil Pritam & Mondal Kartick Chandra. (2022). Frequent itemset mining using FP-tree: a CLA-based approach and its extended application in biodiversity data. Innovations in Systems and Software Engineering,19(3),283-301.

[28] Yang Yang,Na Tian,Yunpeng Wang & Zhenzhou Yuan. (2022). A Parallel FP-Growth Mining Algorithm with Load Balancing Constraints for Traffic Crash Data. INTERNATIONAL JOURNAL OF COMPUTERSCOMMUNICATIONS & CONTROL,17(4),

[29] Chenxiang Xiao,Chenchen Zhang,Bin Zhang,Hui Xu & Hong Liu. (2024). Two-level parallel load balancing strategy for accelerating DSMC simulations in near-continuum gases. International Journal of Modern Physics C,36(03),