Journal of computer science and software applications
https://www.mfacademia.org/
ISSN:2377-0430
Vol. 4, No. 1, 2024

An Efficient Parallel FP-Growth Algorithm for Big Data Association Rule Mining

John Michael Smith

Texas A&M University

Johnms0@gmail.com

Abstract:

The FP-Growth algorithm is a proficient method for correlation analysis, notable for its ability to operate without generating candidate sets and requiring only two database scans. In practical applications, it can represent transaction data within a compressed FP-Tree in memory. This approach addresses the limitations of the Apriori algorithm by decreasing the number of database scans and reducing the candidate sets. However, when applied to large datasets, the FP-Growth algorithm can lead to an excessively large tree structure, straining memory capacity.To address this issue, this paper introduces a parallel FP-Growth algorithm tailored for a big data framework, known as the MRFPG algorithm. This algorithm incorporates load balancing and utilizes intermediate agents to distribute transaction data fragments across a computer cluster. Each node then initiates a map-reduce process to filter out non-frequent items and identify all frequent patterns. Compared to the traditional FP-Growth algorithm, the MRFPG algorithm significantly reduces I/O overhead. Experimental results demonstrate that the MRFPG algorithm is both efficient and rapid.

Keywords:

Big Data, Association Rules, Parallel computing.

1. Introduction

Data mining, also known as the knowledge discovery in the database (KDD), was first proposed at the Eleventh International Joint Academic Conference on artificial intelligence in 1989[1]. Through continuous development, people define it as a process of finding valuable information and knowledge from mass data. In a simple way, it is in a way that a data model of interest to the user is found in a mass of data. In data mining, there is a very important research topic of association rule mining, and association rule mining can find possible in massive data[2]. It means that in a thing database D, we can get the interesting correlation between data and project in terms of computing the degree of support and confidence of transactions. Those sets that support and confidence are larger than user set initial threshold are the association rules we are looking for.

There are two key steps in association rules, one is the discovery of frequent itemsets and the other is the discovery of association rules. Apriori and FP-Growth algorithms are often used when dealing with single - dimensional association rules. The two algorithms are based on serialized single machine algorithm. When data volume reaches a certain level, the computation efficiency is very low, which takes up a lot of I/O resources and costs. At present, there are some scholars to study the efficiency of the improved algorithm, some new research has been proposed as a kind of improved FP-Growth algorithm proposed by Zeng et al in the literature, the algorithm tries to remove the two fork tree steps, although it reduces computer I/ O overhead, but reduces the accuracy of algorithm[3]. There are also some scholars put forward a two-dimensional table based on improved FP-Growth algorithm[4], the algorithm is from a certain extent, improve the efficiency of the original algorithm, reduce the number of scanning database, but there is a significant defect in two-dimensional table. When two key candidates have the same key candidates, that is, if any two rows in the table are the

same, they will not be able to be represented by two-dimensional tables, which also leads to the limitation of the algorithm. In 2015, Rathi and Sheetal proposed the implementation of algorithm using multiple GPU XML data FP-Growth algorithm[5], which reduced the utilization rate of GPU to a certain extent, but under the condition of too much data, requirements for the operation environment was too high.

To solve the above problems, this paper proposes an improved FP-Growth algorithm (proposed parallel load balancing algorithm FP-Growth --MRFPG algorithm based on big data framework), the algorithm set up an intermediate layer to count the idle servers, put the idle servers into a set, and distribute the tasks to the idle servers sequentially. The number of each distribution is determined by the performance of each server, and the threshold can be set in advance. After distributing data to the server cluster, the map and reduce programs are used to calculate by each server separately. The final result is aggregated to the main node of the cluster, and the final frequent itemsets are obtained.

2. Related description

2.1 MapReduce Computational framework

MapReduce is a computing framework proposed by Google in 2004, which is actually a programming mode that can process massive data in parallel[6]. When dealing with a task, MapReduce needs to divide the task process into two steps: map and reduce. The key value pair is the manifestation of input and output at every stage. The working principle of Mapreduce is setting a large data set into small data set and "divide and rule" based on a certain strategy, then only the programmer defined map function and reduce function, map is responsible for processing the data of each block, reduce is responsible for the specification of summary of the calculated results[7], you can get the result you want finally, MapReduce specific working principle as shown in figure 1.

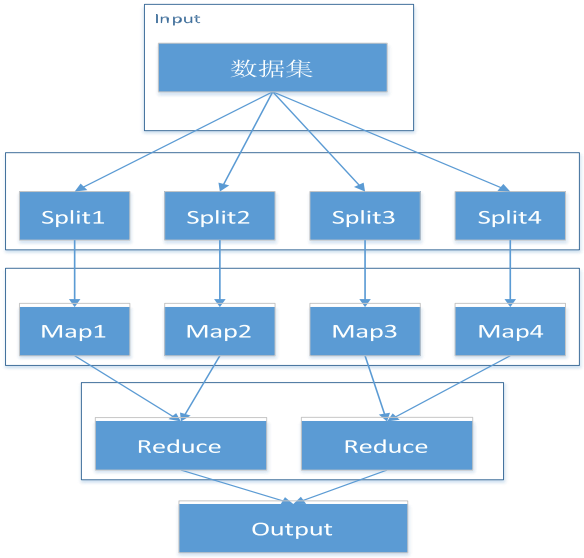

Figure 1 MapReduce execution flow chart

2.2 Basic concepts of association rules

Association rule mining is the discovery of user interested models from massive data. Through this model, we can find some associated between seemingly unrelated data in databases[8]. The most classic case of association rules is the beer and diapers shopping case, this case is a story from the WAL-MART Supermarket: WAL-MART has analyzed the last year's shopping data in a data warehouse and was surprised to find that the goods bought with the diapers were beer[9]. According to this discovery, WAL-MART supermarket has adjusted the layout of the goods and gained a huge profit.

Association rules can be simply expressed as two disjoint sets of A and B in the direct relationship between. Two important concepts of association analysis are support and confidence, Support s refers to the transaction support project set A and the project set B are s% in the transaction database D; Confidence is expressed in C, and C indicates that transactions with c% in D support the project set A while supporting the project set B. Support degree refers to the degree of frequent occurrence of a

transaction in transaction database, it is the yardstick for measuring the scope of association rules. Confidence is the accuracy of a certain association rule, and it is the yardstick for measuring the strength of rules in association rules. Their expressions of support and confidence are shown as follows (1) and (2), where D (X) is the number of transactions of X in the database D, and D is a set of things.

$$\text{Support}(A \rightarrow B) = P(A \cup B) = D(X) / |D| \quad (1)$$

$$\text{Confidence}(A \rightarrow B) = P(B|A) = \text{support}(A \cup B) / \text{support}(A) \quad (2)$$

2.3 FP-Growth algorithm

FP-Growth is a classical association rule mining algorithm is proposed by Professor Han Jiawei in 2000, an efficient mining association analysis algorithm does not generate candidate frequent itemsets, the algorithm by using FP-Tree data structure, the frequent item in the transaction has been compressed to FP-tree in the form of node, prefix the same path and can share this by tailoring the data collection form greatly reduces the overhead of I/O [10], compared with Apriori algorithm, to improve the efficiency of a class.

The FP-Growth algorithm is described as follows:

1) Firstly scan a transaction database to generate 1- frequent itemsets and put them in a List table in descending order.

2) Create a root node and label it as null, then scan the original transaction database again, and get the item set arranged according to the order in List list, then recursively call FP-growth to achieve FP-tree growth.

3) Add a header_table in FP-tree and connect the same item in FP-tree. Find the prefix path (CPB) containing the item from bottom to top in FP-tree.

4) Constructing the condition FP-tree, accumulating the frequent count of item on each CPB, filtering the item below the threshold value, and constructing FP-Tree.

3. MRFPG algorithm

3.1 MRFPG algorithm ideas and steps

The idea of the MRFPG algorithm is to divide it into several partitions based on the size of the original database, and the threshold can be set in advance. Then, we set up an intermediate layer to count the idle servers, that is, servers that have returned to the calculated results. The idle servers are put into a set, and the tasks are distributed to the idle servers in sequence. The number of each distribution is determined by the performance of each server. After distributing data to the server cluster, the map and reduce programs are calculated individually, and the computation is completed by each server individually. Finally, the results are aggregated to the main node of the cluster, and the final frequent itemsets are obtained.

The MRFPG algorithm steps are as follows:

1) Divide the transaction database into several data fragments, which called shard, which are balanced by load and distributed to P nodes.

2) Scan the database for the first time, the use of MapReduce computing framework, removed shard of 1) to mapper, from Input=<key, value=Ti>, output=<key=aj, value=1> (wherein Ti represents each transaction in shard), When all the mapper in the cluster is finished, all the keys on the key=aj assigned to the same reducer, and a summation, eventually get frequent itemsets list List, structure shown in figure 2.

3) The items in List are divided into n groups. They are put into G_list and G_list respectively. Each group is assigned a ID, and each ID contains a set of item sets. The structure diagram is shown in Figure 3.

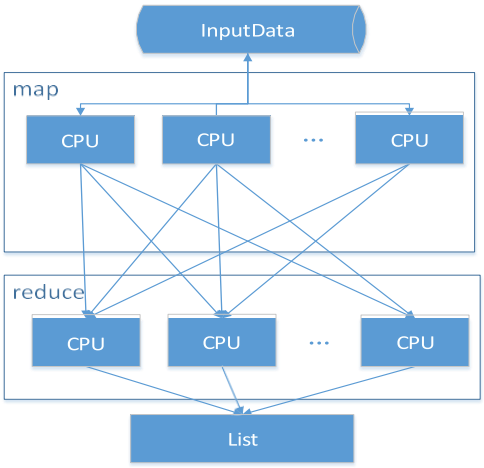

Figure 2 MapReduce processing for Shard

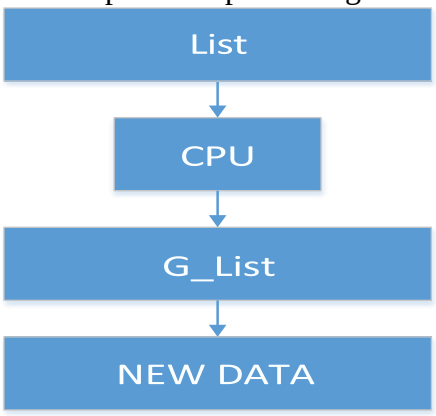

Figure 3 List packet

4) Use MapReduce again. The database is partitioned by mapper, it divides every transaction in the shard database partition in 1) into a single item, and each item according to the G_list mapping to appropriate groups, finally will belong to the same group of data onto the same reducer, finally get the frequent pattern. (<key=group-id , value={{ValueList1} , {ValueList2} , … , {ValueListN}}>→<key=item，reduce ={contains the Top K Frequent Patterns}> of the item, and the structure is shown in Figure 4.

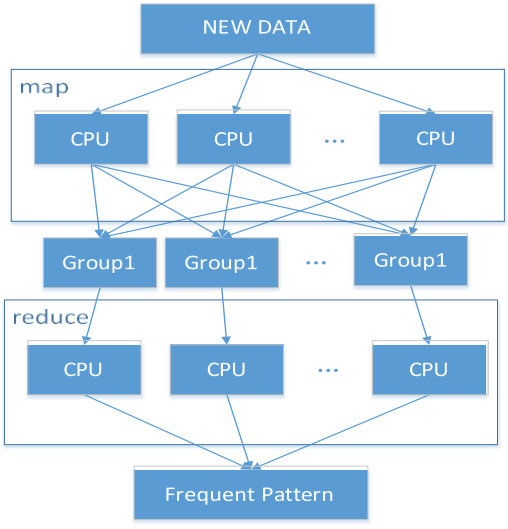

Figure 4 uses MapReduce to get the final frequent itemsets

5) The higher sup frequent pattern is taken as key, and the highest sup frequent pattern containing the key is output. Finally, the results obtained from all mapper are aggregated through reducer, and the output diagram is shown in Figure 5.

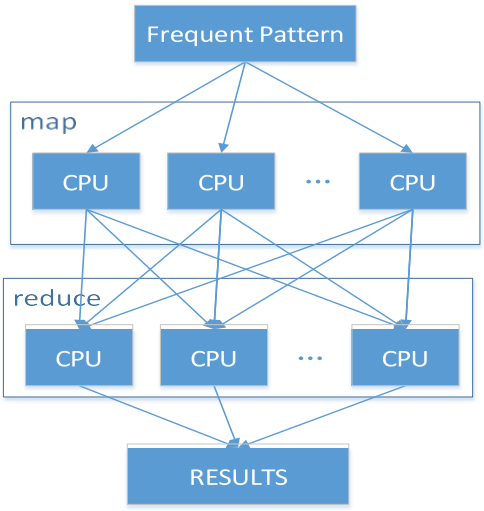

Figure 5 summary results

3.2 MRFPG algorithmic description

Algorithm: MRFPG algorithm

Input: the transaction database D and the minimum support threshold min_sup

Output: all frequent patterns

//Procedure

Load FPGrowthDriver;

Run PFPGrowth;

1. setLoadbalancing();

2.startParallerCounting(){
ParallerCountingMapper;
ParallerCountingReducer;}

3.readFList()&saveFList()

4.startParallerFPGrowth()

ParallerFPGrowthMapper;

ParallerFPGrowthReducer;

5.startAggregating();

AqqreqatorMapper;

AqqreqatorReduce;

//Map

protected void map(LongWritable offset, Text input, Context context) throws IOException, InterruptedException {String[] items = splitter.split(input.toString());

Set<String> uniqueItems = Sets.newHashSet(Arrays.asList(items));

for (String item : uniqueItems) {if (item.trim().isEmpty()) {

continue;}

context.setStatus("Parallel Counting Mapper: " + item);

context.write(new Text(item), ONE)

//Reduce

protected void reduce(Text key, Iterable<LongWritable> values, Context context) throws IOException, InterruptedException {long sum = 0;for (LongWritable value : values) {
context.setStatus("Parallel Counting Reducer :" + key); sum += value.get();}
context.setStatus("Parallel Counting Reducer: " + key + " => " + sum);

```
context.write(key, new LongWritable(sum));}
// Load balancing pseudo code
freeServer = server.setIdentity(id);
server.send(freeServer);
LinkedList<Int> workers = new LinkedList<Int>();
serverItem.recv();
workers.addLast(workerID);
```

3.3 MRFPG algorithm example

The set transactional database is shown in Table 1, the minimum support threshold value is 60%, the minimum support is 5*60%=3, so the item of the support count >=3 is Frequent Item.

Table 1

TID	Data
t1	f,a,c,d,g,I,m,p
t2	a,b,c,f,l,m,o
t3	b,f,h,j,o
t4	b,c,k,s,p
t5	a,f,c,e,l,p,m,n

Suppose there is a computer cluster composed of a master and a slave two machine. Now let them parallelism to calculate the above database and find out all frequent itemsets. According to table 1 and min_sup, we can exclude non frequent 1 itemsets. Finally, we get frequent 1 itemsets of f_1={f:4, c:4, a:3, b:3, m:3, p:3}, scan the database again, delete the non frequent items, and arrange them in descending order of support, and get data as shown in Table 2.

Table 2

TID	Data
t1	f,c,a,m,p
t2	f,c,a,b,m
t3	f,b
t4	c,b,p
t5	f,c,a,m,p

After scanning the database to get Table 2, the transaction in f_1 is divided into two groups according to the transaction location. G_list={{G1: (f:4), (c:4), (a:3)}, {G2: (b:3), (m:3), and (p:3)}}.maper are first derived from the third steps of the algorithm step. The form of the Hash table is shown in Table 3.

Table 3

Key	Value
f	G1
c	G1
a	G1
b	G2
m	G2
p	G2

The transaction is split into a single item array according to the fourth steps in the algorithm step, and the data is shown as shown in Table 4.

Table 4

Item array	Data
a1[]	{f,c,a,m,p}
a2[]	{f,c,a,b,m
a3[]	{f,b}

a4[]	{c,b,p}
a5[]	{f,c,a,m,p}

After getting the item array, according to the MapReduce computing framework, put a1, a2 and a3 into a shard, a4 and a5 into a shard. According to step 4, see which group belongs to each group in the transaction T and send it to the corresponding group. According to the FP-Tree bottom-up traversal method, traversing each group of elements, the first item to get is p, look at the Hash table, p belongs to G2, so it will output with G2, and delete all item that belongs to G2 in the hash table. At this time, Hash table has only the G1 belonging to G1. Then traversing to m, it is found that the G2 of the M item has been sent, so there is no need to send again and return to the continued traversal. In the same way, continue to traverse all items until the Hash table is empty. After that, reducer is processed according to the results of maper, and the input of each reducer is the set of transactions in each Group. Finally, in each machine for processing the FP-Tree data structure on the distribution of data to generate native LFP-Tree (due to space issues, not detailed details), finally to find all frequent item {{f}: 4, {c}: 4, {a}: 3, {b}: 3, {m}: 3, {p}: 3. {f, c}: 3, {f: 3, a}, {f, m}: 3{c, a}: 3, {c: 3, m}, {c, p}: 3, {a: 3, m}, {f, C, a}: 3, {f, C, m}: 3, {f, a, m}: 3. {c, a, m}: 3, {f, C, a, m}: 3}.

4. Experimental results and analysis

In this paper, the MRFPG algorithm is compared with the AFIM algorithm, and the AFIM algorithm is realized by using Eclipse.

Experimental environment: Intel (R) Core (TM) i7-6700HQ, 3.50GHz master frequency, 16G installed memory, 1T hard disk space, 64 bit window7 operating system. The computer are equipped with VMware Workstation Pro virtual machine, and installed a master and a slave CentOS operating system, to build a Hadoop cluster, including 1 NameNode and 8 DataNode, and the configuration of the Hadoop runtime environment, Hadoop version is 2.5.2, JDK version is 1.7.0. Sales data for the experimental data are from a supermarket in October 2007, each record represents a consumer shopping list, transaction data D represents the list of all the shopping, consumers every time shopping on behalf of a transaction, consumers buy goods is a transaction.

This experiment is to test the MRFPG algorithm through the MapReduce computing framework in Hadoop. By load balancing, we allocate different number of node clusters to operate different size data sets and mine frequent itemsets. In the experiment, two data sets, T1 and T2, were allocated randomly. T1 contained 9691 data, and T2 contained 15273 data. The number of nodes per cluster increased by 2, so as to observe the relationship between the number of nodes and computation efficiency. Finally, the experimental results of T1 and T2, as shown in Figure 6 and Figure 7, show that the mining efficiency will increase with the increase of the number of nodes in the Hadoop cluster. Then, we compare the MRFPG algorithm to deal with the total time of transaction execution and the execution time comparison of AFIM algorithm to deal with this transaction by setting different minimum support degrees. The result is shown in Figure 8. At the end of the experiment, the scalability of each algorithm is compared. The comparison results show that the efficiency and scalability of the distributed parallel processing MRFPG algorithm based on big data is obviously higher than that of the AFIM algorithm.

Figure 6 T1 parallel computing time

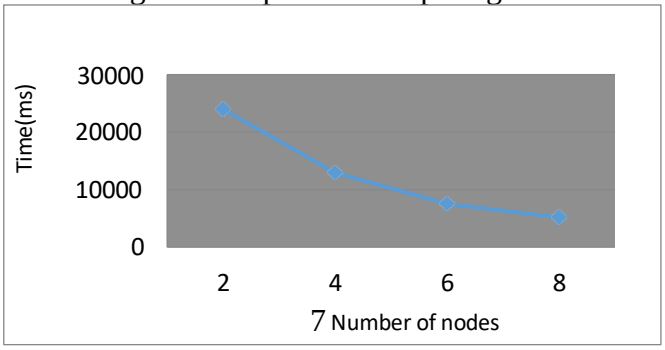

Figure 7 T2 parallel computing time

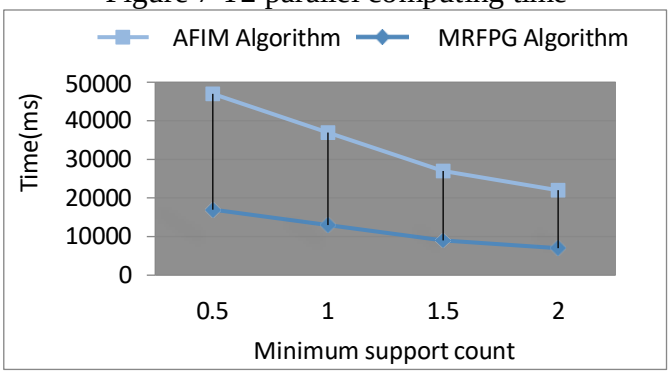

Figure 8 execution time of MRFPG algorithm

5. Conclusion

This paper initially addresses the limitations of the FP-Growth algorithm when handling large-scale data mining. It then elucidates the operational principles of the MapReduce computing framework. Following this, the paper introduces a parallel MRFPG algorithm, designed within a big data framework, providing a comprehensive explanation and illustration.

The MRFPG algorithm leverages load balancing distributed technology and the FP-tree data structure to swiftly extract frequent itemsets from extensive transaction data. Its mining efficiency improves proportionally with the number of nodes in a Hadoop cluster. Experimental results indicate that the MRFPG algorithm outperforms the AFIM algorithm in efficiency and demonstrates excellent scalability.

Acknowledgements

This research is supported by the research fund for humanities and social sciences of the ministry of education under grant No.19XJA910001 and the humanities and social sciences research project of chongqing education commission under grant No. 18SKGH099.

References

[1] Han J W, Micheline K. Concept and technology of data mining[M]. Beijing: Machinery Industry Press, 2013.
[2] Cui Yan, ZQ Bao. Summary of association rules [J]. Computer application research, 2016, 33(02):33-37.
[3] Zeng,Y,Yin S,Liu J,Zhang M. Research of improved FP-growth algorithm in association rules mining[J]. Scientific Programming, 2015, 46(10),281-285.
[4] Yan Yun, YX Luo. Improvement of FP-Tree algorithm [J]. Computer engineering and design, 2010,31(07):1506-1510.
[5] Rathi,Sheetal,Dhote,C.A. Parallel implementation of FP growth algorithm on XML data using multiple GPU[J]. Advances in Intelligent Systems and Computing, 2015.8, 339,581-589.
[6] Tsai,Chih-Fong. Big data mining with parallel computing:A comparison of distributed and MapReduce methodologies[J]. Journal of Systems and Software, 2016,122,83-92.
[7] DEAN J,GHEMAWAT S. MapReduce:simplified data processing on large clusters[J]. Communications Of The ACM, 2008,51(1):107-113.
[8] SH Li, ZW Lv, DY Che. Maximum frequent itemset mining algorithm based on ordered FP-tree[J]. Northeast Normal University newspaper (NATURAL SCIENCE), 2016,48(2):65-69.
[9] Pang-NingT, VipinK, Michael S. Introduction to Data Mining[M]. Beijing:People's post and Telecommunications Press, 2011.
[10] LJ Zhou, X Wang. Research on association rules algorithm in cloud environment[J]. Computer engineering and design, 2014,35(02):499-503.