Improved Balanced Parallel FP-Growth with MapReduce

Qing YANG\textsuperscript{1,a}, Fei-Yang DU\textsuperscript{2,b}, Xi ZHU\textsuperscript{1,c}, Cheng-Gong JIANG\textsuperscript{*}

\textsuperscript{1}Department of Computer Science and Technology, Central China Normal University, Wuhan, Hubei, China
\textsuperscript{2}Collaborative & Innovative Center for Educational Technology, Central China Normal University, Wuhan, Hubei, China
\textsuperscript{a}yangqing@mail.ccnu.edu.cn, \textsuperscript{b}feiyangdu@mails.ccnu.edu.cn, \textsuperscript{c}ssisse@mails.ccnu.edu.cn
\textsuperscript{*}Corresponding author

Keywords: Balanced Grouping, Parallel Computing, FP-Growth, MapReduce.

Abstract. The existing parallel FP-Growth algorithm have solved the problems such as the partition of transaction dataset, which can guarantee that each transaction dataset is independent after the partition, but there are still many problems such as too many iterations in the process of FP-tree mining on single node and low efficiency. What’s more, it did not consider the load balance when the master node divides dataset to the child nodes. By using Cutting strategy on PFP which is the original algorithm with MapReduce, we merged paths which are not frequent in FP-tree, and designed a new parallel FP-Growth algorithm. In addition, the load balancing strategy is used when the master nodes divide dataset to the child nodes. Through the combination of these two strategies, this paper designed a new parallel FP-Growth algorithm.

Introduction

With the arrival of the era of big data, the data set is often TB or even PB level. Computation and I/O volume is very huge when the association rules algorithm deals with massive data sets. Serial algorithms could not afford it while existing association rule mining algorithms are mostly memory consumed algorithms. When dealing with massive data, there is not enough memory or too long time to get a satisfactory result. So the parallelization of association rules mining algorithm came into being.

At present, MapReduce \cite{1} technology is flexible, efficient and virtual, which provides a new solution to solve the problems when processing massive data sets \cite{2}. And the efficiency of traditional frequent item sets mining algorithms, such as FP-Growth \cite{3}, is extremely low. To this end, the parallel computing has been proposed to reduce the memory and CPU consumption. PFP algorithm used MapReduce or Spark framework \cite{4} to implement the parallel processing of mining tasks. By taking the distributed advantages of the Hadoop framework, the algorithm has greatly improved parallel computing. However, in the process of data distribution, PFP algorithm did not consider the problem of balanced grouping, while balanced grouping is very important in dealing with massive data.

Parallel Design of FP-Growth Algorithm

FP-growth Algorithm

The basic idea of FP-growth algorithm \cite{5} is to use tree structure to compress the transaction, and keep the relationship between the attributes in the transaction. The algorithm requires scanning the transaction set twice: The first scan selects frequent items which are then sorted by frequency in descending order to form F-list. The second scan firstly takes the ‘null’ as root node to construct FP-Tree. FP-Growth creates small FP-Tree from the conditional pattern base and executes FP-Growth on the FP-Tree. The process is recursively iterated until no conditional pattern base can be generated.
The FP-Growth Algorithm Based On Cutting Strategy

We designed a frequent item sets mining algorithm based on cutting strategy, Cutting FP-tree algorithm. For the multiple paths item sets in FP-tree, if one path is not frequent and it meet the condition of the above ideas, we can cut the FP-tree. Compared with the original algorithm, the algorithm reduces the number of iterations of the FP-tree processing and improves the efficiency of the algorithm.

After using cutting strategy to construct the FP-Tree, the following is a detailed procedure for Growth mining of the cutting FP-tree:

First step: Determine whether the FP-tree has a single path.

Second step: If the FP-tree has a single path $P$, take all frequent items of the path $P$ as each subset of the set (recorded as $B$). Then the generation mode is $B \cup X$. Its support count is the minimum support count for all nodes.

Third step: Some of the frequent items in FP-tree may exist in multiple paths. For example, frequent item $i$ exists in the prefix path set both $A$ and $B$, $A$ belongs to $B$. The above mentioned cutting strategy can be used to merge the path of set $B$ with the path of set $A$.

Fourth step: Check whether the frequent item $i$ turned into a single path after cutting. If not then continue to the fifth step, or back to the second step.

Fifth step: For each frequent item $a_i$ in the item header table, the generation model is $\beta = a_i \cup x$. The count of the mode is equal to the count of the data item. Conditional frequent pattern tree—Tree$\beta$, is constructed by the conditional pattern base. If Tree$\beta \neq \emptyset$, turn back to the first step until all the frequent pattern sets are mined.

The Improved Balanced Parallel FP-growth Algorithm (IBPFP)

Sharding

While using Hadoop [6], we just need to copy the DB onto Hadoop distributed file system. Hadoop will do the sharding. The default size of each file block is 64M after segmentation. And the size of slice split is determined by the formula $\text{SplitSize} = \max\{\text{minSize}, \min\{\text{maxSize, blockSize}\}\}$.

Parallel Counting

This step uses one MapReduce job to count all the items. The input of this step is the dataset which stored in a distributed file system. The output of this step is the F-List, a list of frequent items sorted by frequency in descending order.

Balanced Grouping

Balanced Grouping fairly divides all the items in the F-List into Q groups, balancing load among all groups, in order to improve parallelization of the whole mining process. Balanced grouping needs two steps.

Load Estimation of Frequent Items. All of the nodes that execute frequent item mining are added together to get the whole load of the parallel FP-Growth process. We can sum up the load of conditional pattern base of all frequent items on a node to obtain the whole load of the node. But it is not possible to get the exact load of the node. Because the load on the basis of the condition pattern base of each frequent item is different according to the different input data. For the above problem, we made the following estimation: The number of recursive iterations during FP-Growth execution on conditional pattern base of each item is estimated as load of FP-Growth on conditional pattern base of each item. The location of each item in F-List is estimated to be the length of the longest frequent path in the conditional pattern base. And the number of recursive iterations is exponentially proportional [7] with longest frequent path in the conditional pattern base. Set frequent item $i$ in Flist position as $P_i$. And set the load it brings as $L_i$, According to the above ideas, we get the Eq. 1:

$$L_i = \log P_i.$$  \hspace{1cm} (1)
**Load Balancing Strategy.** From the Eq. 1, we can know the pair value of each frequent item position in the Flist is equal to the value of its load. The whole question is equivalent to that, split a descending array into Q groups while equalizing the sum of the data in each group as much as possible. So the frequent items in Flist can be grouped by the following method: First, the Q data items in Flist were randomly placed into Q groups, Then repeat the following steps until all the data items in the Flist are assigned.

1. Find group G which sum is smallest in Q.
2. Increase load of group G by load of the new item.

The implementation of the balanced grouping is to save the sum of Q groups load in a minimum heap. Because the load sum of the corresponding group of the heap top is minimum. We only need to add the load of each new data item to the heap top and update it.

**Parallel Cutting FP-tree Algorithm**

This step requires one MapReduce process. First, in the map phase, transactions in original DB are converted into the new group-dependent transactions. Then, in the reduce phase, FP-Tree construction and FP-Growth are recursively done on the group-dependent transactions by each reducer instance.

**Experiments**

We implemented PFP (No parallel improved FP-Growth algorithm), BPF (only balancing strategy is added on the basis of PFP) and IBPFP. Through the analysis and comparison of the data processing time of the three algorithms, we demonstrated the effectiveness of the algorithm IBPFP designed in this paper. The data set used in the experiment is made up of user attention and comment information data of Sina Twitter. In total, there are 185310 transaction records and 68400 data items. The average length of transaction is 300. The longest length of transaction is 2378.

First of all, we compared the processing efficiency of the three algorithms in the same minimum support threshold of 10 and different transaction volume. The comparison is shown in Fig.1

![Figure 1. The processing efficiency in different transaction volume.](image)

Secondly, we compared the processing efficiency of the three algorithms in the same transaction volume of 10000 and different support threshold. The comparison is shown in Fig.2.
Fig. 1 shows that when the number of transactions is relatively small, the processing time of the three algorithms is roughly equal. However, with the increase of transaction volume, IBPFP algorithm began to show the advantages in processing efficiency. The efficiency of IBPFP algorithm has been always higher than BPFP, which proves the effectiveness of the cutting strategy. The efficiency of BPFP algorithm has been always higher than PFP, which proves the effectiveness of the load balancing strategy. However the efficiency difference between IBPFP and BPFP is not very sharp, which proves the improvement of load balancing strategy maybe more obvious than the cutting strategy.

Fig. 2 shows that in condition of the same transaction amount, the minimum support threshold is higher, the processing time of the algorithm is shorter. The above three algorithms are all satisfied with this rule. Compared with BPFP, IBPFP has the cutting process of FP-Tree, which makes some not frequent items paths merged and reduces iteration numbers of part of branches. Compared with PFP, IBPFP has the load balancing process, which makes each node balance loaded. Therefore the IBPFP is better than the BPFP and PFP in different support threshold.

Though the above two experiments, we proved the better efficiency of the IBPFP.

Conclusions and Future Work

In this paper, we studied the MapReduce distributed computing framework and FP-Growth parallel algorithm. We improved the original algorithm by the FP-tree cutting strategy and load balance strategy. And though the experiments, we proved the better efficiency of the new algorithm named IBPFP.

Now the data on the Internet is real-time dynamic growing. How to carry out parallel frequent item mining under this condition will be a new research hotspot.

Acknowledgement

This research was financially supported by Specific funding for education science research by self-determined research funds of CCNU from the colleges’ basic research and operation of MOE.

References


