IPFP: An Improved Parallel FP-Growth Algorithm for Frequent Itemsets Mining

Dawen Xia[1, 2, 4], Yanhui Zhou[1, 2], Zhuobo Rong[1], and Zili Zhang[1, 2, 3, 5]
[1] School of Computer and Information Science, Southwest University, Chongqing, China
[2] Institute of Statistics, Southwest University, Chongqing, China
[3] School of Information Technology, Deakin University, Victoria, Australia
[4] Guizhou Minzu University, Guiyang, China
[5] Corresponding author: Zili Zhang, e-mail: zhangzl@swu.edu.cn

Abstract

As an important part of discovering association rules, frequent itemsets mining plays a key role in mining associations, correlations, causality and other important data mining tasks. Since some traditional frequent itemsets mining algorithms are unable to handle massive small files datasets effectively, such as high memory cost, high I/O overhead, and low computing performance, we propose an improved Parallel FP-Growth (IPFP) algorithm and discuss its applications in this paper. In particular, we introduce a small files processing strategy for massive small files datasets to compensate defects of low read/write speed and low processing efficiency in Hadoop. Moreover, we use MapReduce to implement the parallelization of FP-Growth algorithm, thereby improving the overall performance of frequent itemsets mining. The experimental results show that the IPFP algorithm is feasible and valid with a good speedup and a higher mining efficiency, and can meet the rapidly growing needs of frequent itemsets mining for massive small files datasets.

Keywords: Frequent itemsets mining, Hadoop MapReduce, Parallel FP-Growth, Small files problem

1. Introduction

As data management and analysis are facing new challenges in the age of big data, the rationality and timeliness of the data processing methods are becoming the research hotspot of big data statistical analysis, and big data association analysis greatly increases the profits of enterprises.

As one of the important research directions of data mining, frequent itemsets mining plays an essential role in mining associations, correlations, causality and other important data mining tasks (Agrawal and Srikant (1994); Brin et al. (1997); Silverstein et al. (2000); Han et al. (2000).), which is a strong impetus to the applications of association rules in markets selection, decision analysis and business management (Liao et al. (2010); Cil (2012).). The existing classical frequent itemsets mining algorithms are breadth-first algorithm Apriori proposed by Agrawal et al. in 1994 and depth-priority algorithm FP-Growth presented by Han et al. in 2000. In order to improve the efficiency of frequent itemsets mining, many researchers have proposed several methods to optimize classical algorithms (Park et al. (1995); Grahne et al. (2000); Tsay et al. (2009); Lin et al. (2011).). Meanwhile, to address the bottleneck of mining performance and reduce memory consumption and computation cost of the machine in the single machine environment, parallel and distributed algorithms are proposed (Agrawal and Shafer (1996); Zaki et al.

(1997); Zaïane et al. (2001); Pramudiono and Kitsuregawa (2003); Tanbeer et al. (2009).). For traditional methods, the applications of corresponding algorithms for frequent itemsets mining in the large-scale datasets will easily cause high CPU consumption, high memory cost, high I/O overhead, low computing performance and other issues.

As typical methods of Hadoop Distributed File System (HDFS) and MapReduce parallel programming model provide a new idea for handling big data. In the frequent itemsets mining for large-scale data, a MapReduce approach of parallel FP-Growth (PFP) algorithm is proposed in (Li et al. (2008)), and the performance of PFP algorithm is enhanced by adding load balancing features in (Zhou et al. (2010)), but these methods ignore frequent itemsets mining for massive small files datasets in Hadoop.

With the arrival of big data era, massive data are growing rapidly. However, in reality, most of the large-scale data are composed of massive small files. Small files usually refer to those file sizes, which are less than 64 MB. According to a study in 2007 at the National Energy Research Scientific Computing Center, 43% of the over 13 million files on a shared parallel file system are under 64 KB and 99% are under 64 MB (Petascale Data Storage Institute (2007)), and more scientific applications consist of a large number of small files are depicted in (Carns et al. (2009)). Nevertheless, in the face of massive small files datasets, the constructed FP-tree in Parallel FP-Growth (PFP) algorithm cannot fit into the memory, which often causes problems such as memory overflow and huge communication overhead. Meanwhile, the computing efficiency of the Hadoop platform largely depends on the performance of HDFS and MapReduce (White (2012)), and Hadoop was, at first, designed specifically to handle streaming large files, so when dealing with massive small files, there are significant limitations. Massive small files will reduce the performance of Hadoop, which is mainly shown in the following two aspects: (1) The access efficiency of HDFS is decreased. (2) The additional overhead of MapReduce is increased.

This paper proposes an improved Parallel FP-Growth (IPFP) algorithm and discusses its applications. We introduce a small files processing strategy in the FP-Growth algorithm, to compensate defects of low read/write speed and low processing efficiency for handling the massive small file datasets in Hadoop, and to enhance the access efficiency of HDFS and reduce the additional overhead of MapReduce. On the other hand, we use MapReduce to implement the parallelization of FP-Growth algorithm, thereby improving the overall performance and efficiency of frequent itemsets mining.

The remainder of this paper is organized as follows. First, the proposed algorithm is described in detail in Section 2. Then, the implementation of IPFP algorithm is depicted in Section 3. Next, the results of several experiments and analysis are presented in Section 4, and the actual application of IPFP algorithm is given in Section 5. Finally, the paper is concluded in Section 6.

2. IPFP algorithm description

In this section, we propose the IPFP algorithm for mining frequent itemsets in massive small files datasets in detail.

(1) Write a small files processing program—*Sequence File*. The *Sequence File* is used to merge all massive small files, which are composed of a large amount of transaction datasets stored in HDFS, into a large transaction data file (transaction database).

(2) Equally divide the transaction database into several sub-transaction databases and

then assign them to different nodes in Hadoop cluster. This step is automatically operated by HDFS, when necessary we can use the balance command enabling its file system to achieve load balancing.

(3) Compute *support* count of each item in the transaction database by MapReduce, and then obtain the set of *I_list* from *support* count in descending order.

(4) Divide *I_list* into *M* groups, denoted as *Group_list* (abbreviated as *G_list*), and assign *group_id* for each group sequentially and each *G_list* contains a set of items.

(5) Complete the parallel computing of FP-Growth algorithm by MapReduce. ① The *Map* function compares the item of each transaction in the sub-transaction database with the item in *G_list*. If they are same, then distribute the corresponding transaction to the machine associated with *G_list*. Otherwise, compares with the next item in *G_list*. Eventually, the independent sub-transaction databases corresponded to *G_list* will be produced. ② The *Reduce* function recursively computes the independent sub-transaction databases generated in step ①, and then constructs the FP-tree. This step is similar to the process of traditional FP-tree generation, but the difference is a size *K* max-heap *HP* which stores frequent pattern of each item.

(6) Aggregate the local frequent itemsets generated from each node in the cluster by MapReduce, and finally get the global frequent itemsets.

3. IPFP algorithm implementation

In this section, we implement the IPFP algorithm, which is mainly composed of four steps as described in the following.

Step 1: Merge massive small files. *Sequence File* is composed of a series of <key, value>, where the key is the name of small files and the value is the content of small files before merging. *Sequence File* exploits three classes—*WholeFileInputFormat*, *WholeFileRecordReader* and *SmallFilesToSequenceFileConverter*, to merge the massive small files into a large file. (1) *WholeFileInputFormat* class: 1) The *isSplitable* ( ) method reloads and returns the value of false, and the purpose is to maintain the input file not to be divided into slices. 2) The *getRecordReader* ( ) method returns a customized *RecordReader*. (2) *WholeFileRecordReader* class: the *FileSplit* is converted into a record, where the *key* of the record is the filename and the *value* is the content of this file. (3) S*mallFilesToSequenceFileConverter* class: Massive small files are merged into a sequential file, and this class contains the *Map* ( ) and the *Reduce* ( ). The input format of data is *WholeFileInputFormat*, while the output format is *SequenceFileOutputFormat*.

Step 2: Compute *I_list*. The complexity of time and space is O (TDBsize/P), (*TDBsize*: the size of transaction database, *P*: the number of parallel MapReduce programs).

Step 3: Generate *G_list* from *I_list*, and complete parallel computing of FP-Growth algorithm. *Map* ( ) judges which *G_list* the item of transactions in the machine belongs to, and then sends this transaction to the corresponding *G_list* machine. In order to avoid sending duplicate transaction, we delete the duplicate entries in the hash table. Each *Reduce* ( ) handles the independent sub-transaction database associated with *G_list*, which creates heap *HP* with a size of *K* for each item in *G_list*.

Step 4: Aggregate local frequent itemsets generated in the previous step from each node, and then we get global frequent itemsets.

4. Experiments and results

In this section, we verify the feasibility, speedup, validity and efficiency of IPFP algorithm by two experiments.

The experimental Hadoop cluster is composed of one Master machine and four Slave machines with Intel Pentium（R）Dual-Core E5700 3.00GHz CPU and 2.00GB RAM. All the experiments are performed on Ubuntu 12.04 OS with Hadoop 0.20.2, Jdk 1.6.0 and Eclipse 3.7.1. The real data from the Frequent Itemset Mining Dataset Repository are used as the experimental data[1], which are processed into three groups of different sizes datasets (Datasets1: 2115 small files, Datasets2: 4281 small files and Datasets3: 8583 small files, and each file is less than 64KB). The feasibility, validity, speedup and efficiency are used to evaluate the overall performance of IPFP algorithm and compare it with the PFP algorithm in the same environment.

4.1. Feasibility and speedup evaluation of IPFP

The experimental results are shown in Figure 1. With massive small files datasets, the IPFP algorithm can normally complete distributed computing and accurately find the frequent itemsets in the MapReduce environment, which shows that the IPFP algorithm is feasible. When Datanodes are increasing gradually, the running time of IPFP algorithm significantly decreases and the processing performance is greatly improved, which shows that the IPFP algorithm has a good speedup.

4.2. Validity and efficiency evaluation of IPFP

The experimental results are shown in Figure 2. When the cluster in the pseudo-distributed environment (only one Datanode) switches to a fully distributed environment (more than two Datanodes), the processing capability of IPFP algorithm is significantly enhanced, which shows that the IPFP algorithm is valid. When Datanodes are increasing gradually, the running time of IPFP algorithm is always less than that of PFP algorithm, which shows that the IPFP algorithm has a higher mining efficiency than PFP algorithm.

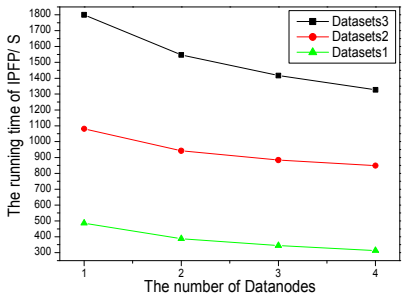
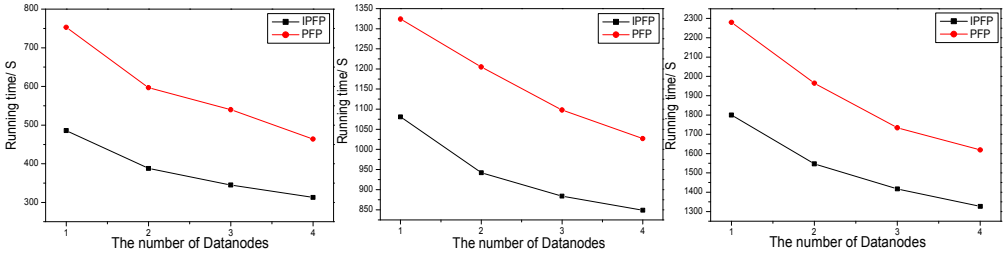


Figure 1. The feasibility and speedup of IPFP.



(a) (b) (c)

Figure 2. The validity and efficiency of IPFP and PFP for frequent itemsets mining in Datasets1 (a), Datasets2 (b) and Datasets3 (c).

[1]http://fimi.ua.ac.be/data/

5. Applications

In this section, we apply the IPFP algorithm to the association analysis of the national college entrance examination and admission data of China.

The IPFP algorithm is used to analyze the unequal strength association of colleges, majors, areas from the real massive small files datasets of the national college entrance examination and admission, which are composed of candidates basic data, recruiting application data, major setting data and admission result data from the year 2003 to 2012 in a province. We have drawn some important and valuable results as follows: (1) There is a strong association relationship among Southwest University, Chongqing University of Posts and Telecommunications, and Southwest University of Political Science and Law; and it is same with Guizhou Minzu University and Guizhou University of Finance and Economics. (2) Colleges lay in the coastal cities have greater popularity, and admission scores of colleges in the North China is much higher than that in the Northwest. (3) The Administration Management major of a college is the most popular, and English and International Economy and Trade major are quite popular as well.

When filling out college recruiting applications, candidates should maintain a certain gradient under the same circumstance, and try to avoid applying to colleges and majors with strong association and colleges in these areas with greater popularity, so that they will obtain more admission opportunities. At the same time, in this association analysis, the IPFP algorithm shows better processing performance and a higher mining efficiency than PFP algorithm.

6. Conclusions

In this paper, owing to the small files processing strategy, the IPFP algorithm can reduce memory cost greatly and improve the efficiency of data access, thus avoids memory overflow and reduces I/O overhead. Meanwhile, the IPFP algorithm is migrated to the MapReduce environment, which can complete frequent itemsets mining efficiently and thus enhance the overall performance of FP-Growth algorithm. The experimental results show that IPFP algorithm can make a breakthrough where PFP algorithm has its defects in handling massive small files datasets, and has a good speedup and a higher mining efficiency.

Acknowledgements This work was supported by the National Science and Technology Support Program of China (No. 2012BAD35B08), the Science and Technology Foundation of Guizhou (No. LKM201212), and the SWU Grant for Statistics Ph.D.

References

Agrawal, R. and Shafer, J. C. (1996) “Parallel mining of association rules,” *IEEE Transactions on Knowledge and Data Engineering*, 8(6), 962-969.

Agrawal, R. and Srikant, R. (1994) “Fast algorithms for mining association rules,” *In: Proceedings of the 1994 international conference on very large data bases (VLDB’94)*, Santiago, Chile, 487-499.

Brin, S., Motwani, R. and Silverstein, C. (1997) “Beyond market basket: generalizing association rules to correlations,” *In: Proceeding of the 1997 ACM SIGMOD international conference on management of data (SIGMOD’97)*, Tucson, AZ, 265-276.

Carns, P., Lang, S., Ross, R., Vilayannur, M., Kunkel, J. and Ludwig, T. (2009) "Small File access in parallel file systems," *In: Proceeding of the 2009 IEEE international symposium on parallel and distributed processing*, Rome, Italy, 1-11.

Cil, I. (2012) "Consumption universes based supermarket layout through association rule mining and multidimensional scaling," *Expert Systems with Applications*, 39(10), 8611-8625.

Grahne, G., Lakshmanan, L. and Wang, X. (2000) "Efficient mining of constrained correlated sets," *In: Proceeding of the 2000 international conference on data engineering (ICDE'00)*, San Diego, CA, 512-521.

Han, J., Pei, J. and Yin, Y. (2000) "Mining frequent patterns without candidate generation," *In: Proceeding of the 2000 ACM SIGMOD international conference on management of data (SIGMOD'00)*, Dallas, TX, 1-12.

Li, H., Wang, Y., Zhang, D., Zhang, M. and Chang, E. Y. (2008) "PFP: Parallel FP-Growth for query recommendation," *In: Proceeding of the 2008 ACM conference on Recommender systems,* Lausanne, Switzerland, 107-114.

Liao, S., Chen, Y. J. and Deng, M. (2010) "Mining customer knowledge for tourism new product development and customer relationship management," *Expert Systems with Applications*, 37(6), 4212-4223.

Lin, K. C., Liao, I. E. and Chen, Z. S. (2011) "An improved frequent pattern growth method for mining association rules," *Expert Systems with Applications*, 38(5), 5154-5161.

Park, J. S., Chen, M. S. and Yu, P. S. (1995) "An effective hash-based algorithm for mining association rules," *In: Proceeding of the 1995 ACM-SIGMOD international conference on management of data (SIGMOD'95)*, San Jose, CA, 175-186.

Petascale Data Storage Institute. (2007) "NERSC file system statistics," *World Wide Web electronic publication*, Online, Available: http://pdsi.nersc.gov/filesystem.htm.

Pramudiono, I. and Kitsuregawa, M. (2003) "Parallel FP-Growth on PC Cluster," *Advances in Knowledge Discovery and Data Mining*, 2637, 467-473.

Silverstein, C., Brin, S., Motwani, R. and Ullman, J. (2000) "Scalable techniques for mining causal structures," *Data Mining and Knowledge Discovery*, 4(2-3), 163-192.

Tanbeer, S. K., Ahmed, C. F. and Jeong, B. S. (2009) "Parallel and distributed algorithms for frequent pattern mining in large databases," *IETE Technical Review*, 26(1), 55-65.

Tsay, Y. J., Hsu, T. J. and Yu, J. R. (2009) "HUT: A new method for mining frequent itemsets," *Information Sciences*, 179(11), 1724-1737.

White, T. (2012) *Hadoop: The Definitive Guide*, *3rd ed.*, O'Reilly Media Inc, Sebastopol, CA.

Zaïane, O. R., EI-Hajj, M. and Lu, P. (2001) "Fast parallel association rules mining without candidacy generation," *In: Proceeding of the 2001 international conference on data mining (ICDM'01)*, San Jose, CA, 665-668.

Zaki, M. J., Parthasarathy, S., Ogihara, M. and Li, W. (1997) "Parallel algorithm for discovery of association rules," *Data Mining and Knowledge Discovery*, 1(4), 343-373.

Zhou, L., Zhong, Z., Chang, J., Li, J., Huang, J. Z. and Feng, S. (2010) "Balanced parallel FP-Growth with MapReduce," *In: Proceeding of the 2010 IEEE Youth conference on information computing and telecommunications (YC-ICT'10)*, Beijing, China, 243-246.