Research of the FP-Growth Algorithm Based on Cloud Environments

Lijuan Zhou
Information Engineering College, Capital Normal University, Beijing, China
Email: zlj87@139.com

Xiang Wang
Information Engineering College, Capital Normal University, Beijing, China
Email: wx3710831988@126.com

Abstract—The emergence of cloud computing solves the problems that traditional data mining algorithms encounter when dealing with large data. This paper studies the FP-Growth algorithm and proposes a parallel linked list-based FPG algorithm based on MapReduce programming model, named as the PLFPG algorithm. And then it describes the main idea of algorithm. Finally, by using different data sets to test the algorithm, the experimental result shows that PLFPG algorithm has higher efficiency and better flexibility and scalability.

***Index Terms*—Cloud computing, FP-Growth, MapReduce, Data mining**

I. Introduction

Association rules are one of the most active research methods in data mining. Association rules mining is to find strong association rules. That can be divided into two sub-problems, discovering frequent item-sets and generating association rules. The salient features of data mining technology are that it discovers implicit and useful knowledge from huge, complex and high-latitude data sets. This puts special challenges on association rules techniques.

In 1994, Agrawal proposed the famous Apriori algorithm, but there are two drawbacks in it. First, because it repeatedly scans the transaction database, it needs a lot of I/O load; Second, it will cause huge candidate set. FP-Growth is a good solution to the above two problems. The biggest advantage of the FP-Growth algorithm is that it only scans database twice. It directly compresses the database into a frequent pattern tree instead of using a candidate set and finally generates association rules through FP-tree.

However, FP-tree and conditional FP-tree structure and traversal consume most of the time in the FP-Growth algorithm. There are two problems. First, for each recursive digging, the algorithm must generate a new conditional FP-tree. That affects the efficiency of the algorithm's time and space. This article mines recursively based on the form of a linked list instead of conditional FP-tree. It greatly saves time and space and improves efficiency by the way that linked lists record the relationship among the conditional FP-tree nodes.

Second, in the era of data explosion today, the data sets are very large and the FP-tree has a lot of long branches, the traditional algorithm needs to construct a huge FP-tree. In this situation it has defects, such as a small amount of processing within unit time, the long processing time and so on. It is difficult to achieve the desired effect. For this problem, combining association rule analysis and cloud computing to design efficient parallel algorithms has become an inevitable trend.

Hadoop is a distributed computing framework developed by the Apache Foundation. Users can develop distributed programs in the case of not understanding the distributed underlying details [14][15]. The framework makes development and parallel processing of large-scale data easier. Meanwhile it is high reliable, scalable, efficient and tolerant. It has been applied in Amazon, Facebook, Yahoo, IBM and other large sites.

In this article we use the framework to implement the parallel FP-Growth algorithm based on the linked list, PLFPG to discover valuable knowledge model from large-scale data in a fast and efficient way.

The remaining of the paper is organized as follows. Section II describes the related concepts. The main idea of our proposed algorithm is discussed in detail in Section III. Section IV illustrates experimental results and performance analysis. Finally Section V will give the conclusion.

II. Related Concepts and Description

A. Item Sets Space Theory

Agrawal et al. established the item-sets space theory for transactional database mining. The core principle of this theory is that the subsets of frequent item-sets are frequent item-sets; the superset of non-frequent item-sets is non-frequent [6]. This principle, anti-monotone property, has been applied as a classic data mining theory. In 1994, they proposed Apriori and the Apriori algorithm has been still widely discussed as the classic association rule mining algorithm. But with further research, its shortcomings exposed. For every k cycles, the algorithm

doi:10.4304/jsw.9.3.676-683

has to scan the database once to verify it whether or not to join Lk for each element of the candidate set Ck. If a frequent itemset contains 15 items, then it needs to scan the database 15 times at least. At the same time it will cause huge candidate set. That is exponential growth. When the number of frequent item sets is large, it will produce a huge candidate set. This is a challenge for time and memory space.

In order to improve the efficiency of Apriori algorithm, it appears a series of improved algorithms, such as the data partitioning method, hash-based method, transaction compression method and so on. Although they still follow the above theory, due to the introduction of the relevant technology, these algorithms improve the adaptability and efficiency of Apriori algorithm to some extent.

B. FP-Growth Algorithm

Apriori is a classic algorithm for mining frequent item-sets, which has a very important nature: all non-empty subsets of frequent item-sets must are also frequent. But it has to scan database multiply before it produces frequent patterns and at the same time produces a large number of candidate frequent sets. That makes the Apriori algorithm have larger time and space complexity. Besides, the performance of Apriori algorithm in mining long frequent patterns is often low.

In 2000, Han proposed the FP-Growth algorithm. Because it only scans the database twice and does not use the candidate set, the algorithm became very well-known and efficient. The basic idea is that first sweep the transactional database to find frequent 1-item sets, and then construct the FP-tree. At last it discovers conditional pattern base to mine frequent pattern based on the FP-tree [3].

Use transaction database data to construct FP-tree

1) Scan database for the first time, get frequent 1- item sets L.

2) Create the root of the tree with the "root" tag. Scan DB for the second time and create a branch for each transaction.

Mining frequent patterns from the FP-tree

1) For each item, generate its conditional pattern base and then its conditional FP-tree;

2) For each new generated conditional FP-tree, repeat this step until the FP-tree is null or it only has unique branch;

The algorithm mining frequent patterns in FP-tree is as follows:

Input: the tectonic good FP-tree; transaction database DB; minimum support threshold Minsup.

Output: the complete set of frequent patterns.

Method: Call FP-growth (FP-tree, null).

The core for mining FP-Tree algorithm is the FP-growth process. It achieves frequent patterns by the way of recursive calls.

FP-growth algorithm is as follows,

```
FP-growth (Tree, a)
If (Tree contains only a single path P)
   then for each combination of the junction in the path
      P (denoted by b)
   do
      generating mode bUa, its support = minimum
      support of nodes in b;
else
   for each ai, in the header table of FP-tree (reverse)
   do begin
      produce a model bi = ai U a, its
      support=ai.support;
      construct conditional pattern base for b and then
      construct conditional FP-tree for b, Tree b;
      If Tree b ≠ Ø, then call FP-growth (Tree b, b);
```

Most of the time is consumed on the FP-tree and conditional FP-tree structure and traversal in traditional FP-Growth algorithm. When data sets are very large, time and space efficiency will become very low, even digging failure. PLFPG is a parallel algorithm based on linked list in cloud computing environment, which has a good solution to the performance bottleneck of traditional FP-Growth algorithm. It uses two linked lists to substitute conditional FP-trees constructed in each recursive and saves time and space. At the same time based on the MapReduce programming model for the distribution of large data sets to the individual computing nodes, the FP-tree constructed in each computing node is not very large and so it reduces the requirement for computer hardware.

C. Building a Cloud Platform

This paper builds a cloud platform to implement the parallel FP-Growth algorithm based on the linked list, PLFPG. Hadoop is able to take full advantage of the power of clusters to compute and store tasks in high speed. Hadoop is a software framework which can process large amounts of data distributed and is reliable, efficient, and scalable. Hadoop assumes that computing elements and storage can fail and so it maintains multiple working copies of data to ensure the redistribution process for the failed node. So it is reliable. Hadoop works in parallel and speeds up processing through the way of parallel computing. So it is efficient. Hadoop is scalable and capable of handling the PB level data [16]. Therefore, Hadoop is suitable for the algorithm.

The structure of Hadoop components is shown in Fig. 1. In the architecture, Hadoop Common provides a generic function blocks to support the Hadoop subprojects. MapReduce components provide Map and Reduce processing. HDFS complies distributed file storage mechanism. ZooKeeper provides basic services like distributed lock for building distributed applications [5]. The most core designs of Hadoop are HDFS and MapReduce computation model [4], HDFS is an implementation of Hadoop Distributed File System and provides the underlying support for distributed computing storage. The idea of MapReudce was first raised by one of Google's papers. Simple explanation for MapReduce is that task decomposition and a summary of the results.

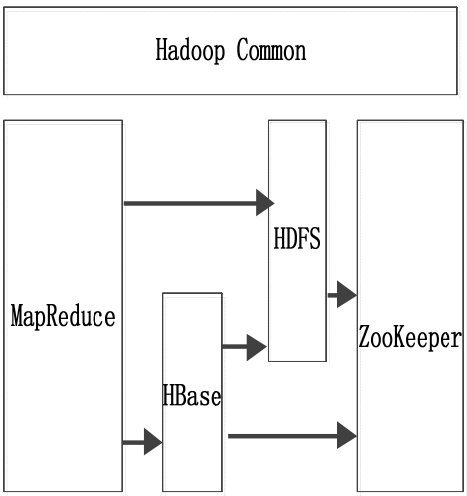


Figure 1. Hadoop components structure

HDFS is a highly fault-tolerant distributed system. Distributed File System has the following basic characteristics:

1) a single namespace for the entire cluster

2) data consistency, suitable for write-once multiple read model. The client can not see the existence of the file before it is not successfully created.

3) the file will be divided into multiple folders. Each file block is allocated to store the data node. It will have to copy the file block to guarantee the security of the data according to the configuration.

HDFS is suitable for deployment in a cheap machine. HDFS provides high throughput data access and ideal for applications on large-scale data sets. HDFS is a master-slave structure system. HDFS clusters are made from a NameNode and many DataNodes. Each node is a common PC.

HDFS has three important roles, NameNode, DataNode, Client.

NameNode can be seen as a manager for the distributed file system. It is primarily responsible for managing the file system's namespace, cluster configuration and storage block replication. NameNode will store Meta-data of the file system in memory. These include the file information, the file blocks information corresponding to each file and the information of each file block in DataNode.

DataNode is the basic unit of the file storage. It stores Block in the local file system and saves only the Meta-data of Block. At the same time it sends reports of all existing Blocks periodically to NameNode.

Client is the application procedure that needs to obtain the distributed file system files.

MapReduce is a programming model for large-scale data set (more than 1TB). Its main idea is borrowed from the functional programming language as well as vector programming language. It greatly facilitates that programmers make their own procedures run in the distributed system without knowing the parallel programming. Fig. 2 shows the approximate data flow diagrams of MapReduce. It is a highly efficient distributed programming model [12]. It divides calculation process into two phases: Map and Reduce. The input of each stage is a series of key-value pairs (key / value), and the output of each stage is also a series of key-value pairs, as follows:

Map: (k1, v1) → list (k2, v2), receives the key-values (k1, v1), after it processes them, user-written Map outputs intermediate key-value pairs (k2, v2). The MapReduce system will merge all intermediate values, output (k2, list (v2)), and transmit to the Reduce method according to the key-value pairs automatically.

Reduce: (k2, list (v2)) → list (k3, v3), receives the key-value pairs (k2, list (v2)) outputted by Map stage. After the user-written Reduce code processes them, these values are merged to form a smaller set of values, and outputted to HDFS.

This paper designs appropriate Combiner function for PLFPG algorithm on the basis above. Combiner is a very important middle function. It reduces the data transmission between map and reduce tasks, saving bandwidth.

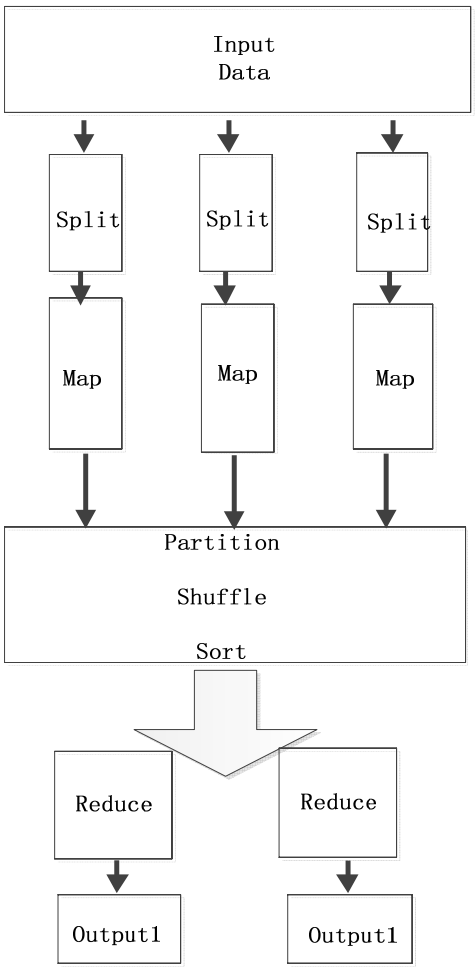


Figure 2. MapReduce approximate data flow diagram

III. MAIN IDEA OF PLFPG ALGORITHM

This paper proposes a parallel FP-Growth algorithm based on a linked list structure named as PLFPG. It mines frequent items based on linked lists. It records the conditional FP-tree structure through the establishment of linked lists instead of constructing conditional FP-tree in each recursive process. That saves time and space.

It is achieved in parallel with Hadoop computing framework [9]. At the same time it considers that tasks are rationally assigned. The idea of the algorithm is as follows.

A. Mining Based on Link List

Traditional FP-Growth algorithm must generate conditional FP-tree every recursive mining process [13]. When recursion depth is very big, it is unavoidable to frequently create sub-FP-tree. This article uses linked lists to record the relationship among the conditional FP-tree nodes. That greatly saves time and space and improves efficiency. Code is as follows:

```
typedef struct branchNodeList
{
int nodeNum;
treeNode* nextTreeNode;
branchNodeList * nextNode;
}branchNodeList;
```

Wherein nodeNum records the number of the branch endpoint item of a subtree (denoted as subTreeA) in recursive mining process. endPointTreeNode points to the branch endpoint item of subTreeA, nextNode points to next branch endpoint item of subTreeA.

```
typedef struct nodeList
{
treeNode treeSnode;
int num;
} nodeList;
```

treeSnode means the endpoint item value of subTreeA, num is the total number of the endpoint item.

As shown in Fig. 3, the article introduces the concept of linked list structure mining. In each recursive mining process it records the information of the context only through the linked list. The time and space efficiency of PLFPG has been greatly improved than that of classic FP-Growth algorithm [1].

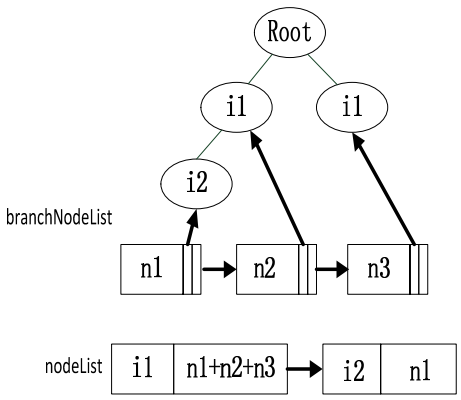


Figure 3. Linked list structure mining

B. Parallel Implementation

Traditional FP-Growth algorithm may fail in the mining of large data sets, because the algorithm calculates the support of various items in massive data support and compresses all the records in the database into a tree. "Big Data" is the data sets in which there is particularly large amount of data and data categories. And such data sets can't be counted, calculated and processed with the traditional database tools, because that requires the computer's computing and storage performance particularly high and the data continues to grow. It is unrealistic using the traditional FPG algorithm to handle massive amounts of data.

Hadoop platform solves this problem well. MapReduce distributes the operations on large data sets to a master node and some sub-node in order to complete it together. Then through integration of the intermediate results of each node it gets the final result. PLFPG algorithm makes full use of various computer resources and reduces the requirements of traditional computer configuration. The following is the thought of parallel Implementation [8].

- **Distributed Statistics the Number of Items**

Table I shows a simple transaction database example. It is divided into a continuous several parts by InputFormat. Each part is a data slice. The data slice is stored in each computer node and then each node performs MapReduce statistical functions to get the number of each item to store in the F-List. As is shown in Fig. 4, we discuss that existing data sets are allocated evenly to three Mapper in the example. The task of each Mapper is responsible for adding up the number of individual items in the Mapper data slice. Combiner intermediate function merges the intermediate results outputted by each Mapper in order to reduce the transmission of data between map tasks and reduce tasks. Then after the MapReduce framework handles it, finally the output data is sent to the reduce function to get the final result. Count the number of each item in the database and store items whose support is greater than or equal to the minimum support in F-List (the support of thesis instance is 3), F-List = {I3: 7, I1: 6, I2: 6, I5: 4, I4: 3}, F-List memories the maximum frequent item sets.

A Combiner intermediate function is mentioned here. The available bandwidth of clusters limits the number of MapReduce jobs, so the most important thing is to try to avoid the transmission of data between map tasks and reduce tasks. Hadoop allows users to specify a merge function Combiner for output of map tasks. Its output is the input of reduce. Merge function is an optimization program. No matter how many times Combiner function is called when the program is running, the final output is consistent with each other [7].

TABLE I.

TRANSACTION DATABASE.

TID	Transaction	TID	Transaction
1	I1 I3 I5	6	I1 I2 I5
2	I1 I2 I3	7	I1 I2 I3
3	I3 I5	8	I1 I3
4	I2 I3 I4	9	I1 I2 I3 I4 I5
5	I2 I4		

The pseudo-code is as follows:

Procedure: Mapper(key, value=Ti)
foreach item ai in Ti do
Call Output(<ai,'1'>);
end

Procedure: Combiner(key,value=Output(<ai,'1'>) of each mapper)
C ←0;
foreach item '1' in ai do
C← C+1;
Call Output(<ai,C>);
end

Procedure: Reducer(key=ai , value=S(ai))
D← 0;
foreach item 'C' in Ti do
D ← D+C;
end
Call Output(<ai,D>);

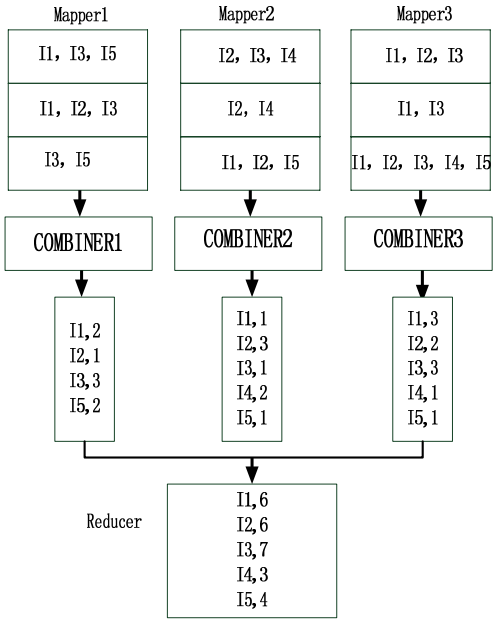


Figure 4. Distributed statistics the number of projects

The transaction database is sorted in accordance with the frequency size of items. The results are shown in Table Ⅱ.

TABLE Ⅱ.

SORT TRANSACTION DATABASE

TID	Transaction	TID	Transaction
1	I3 I1 I5	6	I1 I2 I5
2	I3 I1 I2	7	I3 I1 I2
3	I3 I5	8	I3 I1
4	I3 I2 I4	9	I3 I1 I2 I5 I4
5	I2 I4		

- **Weighing the Allocation of Tasks**

After getting the support of each item, we construct distributed FPtree to mine frequent itemsets. Inappropriate task allocation in a distributed environment will reduce the utilization of resources, seriously affect the efficiency of program execution [10] [11]. If the task is assigned unreasonably, it will cause the following: the execution time of the computer A is T, in a wait state because the task is finished; the execution time of the computer B is 10T. Due to in the distributed environment, it doesn't complete the final overall computing until it integrates the various results after all compute nodes finish the calculation. The execution time of the entire task is 10T. It takes the maximum running time reducing the program's execution speed. If we want to speed up the overall computing speed, we need allocate tasks reasonably. Therefore, parallelizing FP-Growth algorithm and weighing the task split to construct distributed FP-Tree are very important issues to be considered.

The steps of the allocation of tasks are as follows:

1) In the process of executing it, the FP-Growth algorithm has sorted individual transaction records in accordance with the frequent item sets F-List and counted the end of the records sorted to build follow-up FP-Tree. In the data set in Table Ⅱ, the first record is {I1, I3, I5}. After it is sorted, it becomes {I3, I1, I5}. The number of i5 adds 1 (initially 0), and so, the result of the complete data set is expressed as LE-List = {I1: 1, I2: 2, I4: 3, I5: 3}.

2) LE-List is a superset of FP-Tree leaf nodes. We distribute leaf nodes equally. Then records containing the leaf nodes are assigned to the group reducer to build the FP-Tree. For example, for the data set in Table Ⅱ, We can assign them into three tasks to build three sub FP-Tree according to LE-List. I1 and I2 are in a group, I4 a group, I5 a group [2]. Three FP-Tree are shown in Fig. 5. At last we get frequent item sets after finishing FP-growth algorithm based on a linked list mining in each computer node through MapReduce computation model.

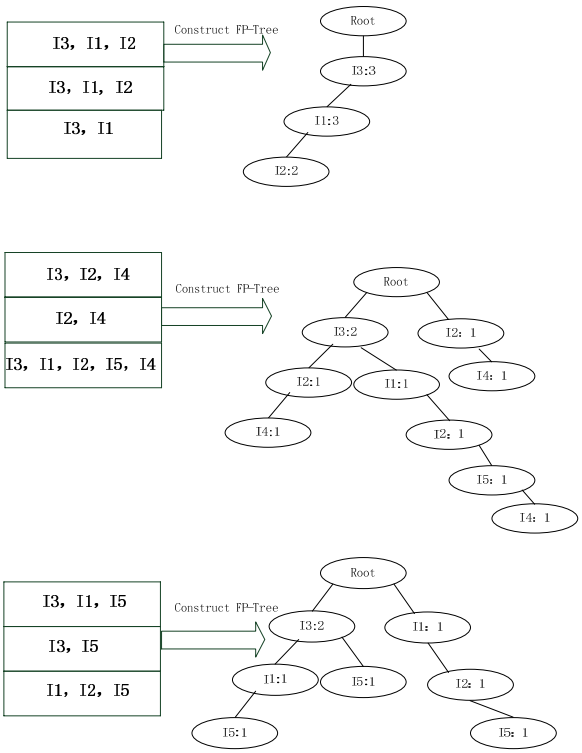


Figure5. Balanced to create a FP-Tree

- **Integration**

Integrate the frequent pattern of the previous step and the output form is <itemi, Frequent pattern set i>. Frequent pattern set i is the collection of all frequent patterns containing this item i. It arranged in descending order according to the support.

IV. Results Analysis

A. Experimental Environment

All experiments of this article were run on the Hadoop platform built in the laboratory. The platform consists of 4 machines, the hardware configuration of these 4 machines are identical , they all have quad-core Intel Corei5 processor, 16G memory. Each node is running Ubuntu Linux operating system. The Hadoop version is 0.20.2, java version is 1.6.25, and between each machine was connected through a switch with a Gigabit Ethernet card.

B. Experimental Data And Results

The experimental data is the network log request on the server. In order to test the performance of the algorithm, the experiments used data containing 100000, 1000000, 2000000 records to test. Transaction records consist of gender, game one, game two, ... , game k, the maximum length of which is 15. To obtain the relationship between gender and games. After running and experimental calculations, the following results were shown in Table Ⅲ and Fig. 6. In Fig. 6, the abscissa is for the amount of data and the ordinate is for the running time, Seen intuitively from Fig. 6, the running efficiency of the PLFPG algorithm, based on the linked list parallel FPG algorithm with MapReduce, in dealing with massive data sets, is much higher than that of the traditional FP-Growth algorithm.

TABLE Ⅲ.

THE OPERATION RESULT OF THE ALGORITHM

model\the number of data	100000	1000000	2000000
traditional FP-Growth	108421ms	486574ms	1002430ms
4 machines running PLFPG	40152ms	121560ms	210373ms
2 machines running PLFPG	50305ms	198778ms	427574ms

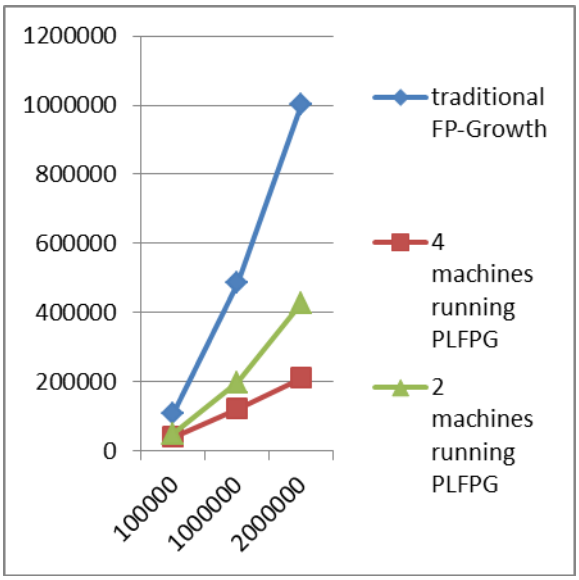


Figure 6. Comparison of running time of algorithm

Before the advent of the huge amounts of data, traditional data mining algorithms can be a good solution to the problem and discover knowledge. But in the face of the massive data, traditional algorithms do not have good scalability and extensibility and can't discover useful knowledge from the huge data quickly and accurately. In Fig. 7, abscissa is for the number of nodes, vertical axis is for the running time and the legend is for the amount of data. It can be seen from Fig. 7, with the increase in the cluster nodes, the efficiency of this algorithm is also increased. This also verified PLFPG algorithm has good scalability and extensibility and can be effectively used for the analysis of large-scale data mining.

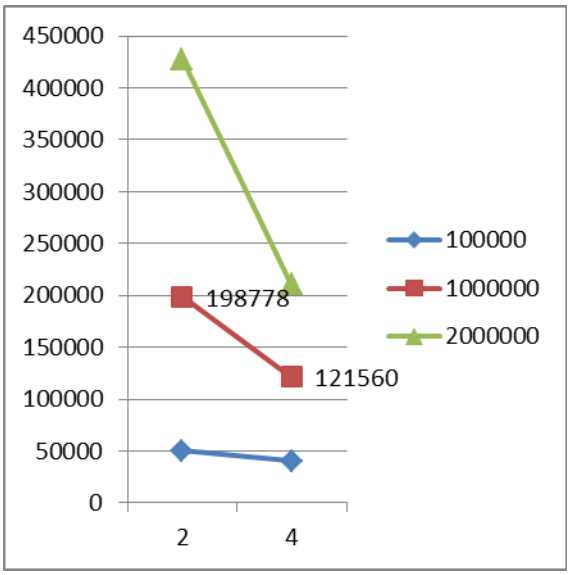


Figure 7. Comparison of running time of algorithm

The speedup is the ratio of the running time consumed by the same task in a single-processor system and the parallel processor system. It is used to measure the performance and effectiveness of the parallel system or program parallelization. Fig. 8 shows the speedup of PLFPG algorithm. It can be seen from the figure, there is a clear acceleration of the algorithm. So it proves the efficiency of the distributed algorithm, PLFPG.

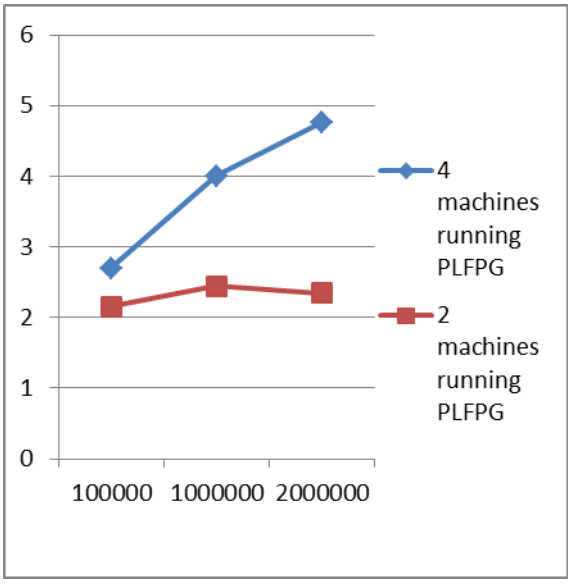


Figure 8. speed up

V. Conclusion

For poor computing power, scalability defects of traditional data mining algorithms in dealing with huge amounts of data, the paper presents a parallel FPG algorithm based on the linked list structure running in MapReduce programming model, named as PLFPG. This algorithm improves the shortcomings of the traditional FP-Growth algorithm. First, it describes item-sets space theory, the basic idea of FP-Growth algorithm and the basic components of the Hadoop platform, including HDFS framework and MapReduce programming model. Then, it describes the PLFPG algorithm design ideas. Finally, the algorithm was validated by varying the size of the data set. The results show that the PLFPG algorithm compared with the traditional FP-Growth algorithm has a higher operating efficiency and better scalability and extensibility. It can effectively analysis and deal with large data sets.

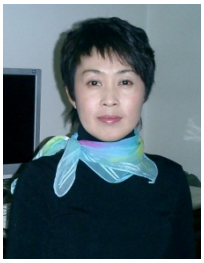
Acknowledgement

This research was supported by China National Key Technology R&D Program (2012BAH20B03), (2013BAH19F01). National Nature Science Foundation (31101078), Beijing Nature Science Foundation (4122016), "The computer application technology" Beijing municipal key construction of the discipline, Beijing Engineering Research Center, and Beijing Educational Committee science and technology development plan project (KM201110028018).

References

[1] Ming Fan, Chuan Li, “Mining Frequent Patterns in FP-tree without Conditional FP-tree”, Computer Research and Development, Vol. 40, No. 8, 2003.
[2] Peng Zhao, “Research Mining Frequent Items Algorithm in Massive High-dimensional Data Sets”, Computer Applications and Software, 2012.
[3] J. Han and M. Kamber, “Data Mining: Concepts and Techniques”, Morgan Kaufmann, 2000.
[4] Liu Peng, “Cloud computing (second edition)”, Beijing: Electronic Industry Press,2011:189-219.
[5] Shulan Zhao, “typical Hadoop cloud computing”, Electronic Industry Press, Beijing, 2013.
[6] Guojun Mao, Lijuan Duan, Shi Wang, Yun Shi, “data mining principles and algorithms (the second edition)”, Tsinghua University Press, Beijing, 2007.
[7] Tom White, “Hadoop: The Definitive Guide, Second Editon”, Tsinghua University Press, 2011.
[8] Haoyuan Li, Yi Wang, Dong Zhang, Ming Zhang, Edward Chang, “Pfp: Parallel Fp-Growth for Query Recommendation”, RecSys '08 Proceedings of the 2008 ACM conference on Recommender systems, Pages 107-114 ACM New York, NY, USA ©2008.
[9] Lamine M. Aouad, Nhien-An Le-Khac, TaharM.Kechadi, “Distributed Frequent Item-sets Mining in Heterogeneous Platforms”, Engineering, Computing and Archtecture, Volume 1, 2007.
[10] Le Zhou, Zhiyong Zhong, Jin Chang, Junjie Li, Huang, J.Z., Shengzhong Feng, “Balanced parallel FP-Growth with MapReduce”, Information Computing and Telecommunications (YC-ICT), 2010 IEEE Youth Conference on, Page(s): 243 -246 ,2010.
[11] Yang Liu, Maozhen Li, Alham, N.K., Hammoud, S. , Ponraj, M. “Load balancing in MapReduce environments for data intensive applications”, Fuzzy Systems and Knowledge Discovery (FSKD), 2011 Eighth International Conference on,Page(s): 2675 - 2678 ,2011.
[12] Jeffrey Dean, Sanjay Ghemawat, “MapReduce: simplified data processing on large clusters”, Communications of the ACM - 50th anniversary, Volume 51 Issue 1, Pages 107-113,2008.

[13] B.Santhosh Kumar, K.V.Rukmani, “Implementation of Web Usage Mining Using APRIORI and FP Growth Algorithms”, Int. J. of Advanced Networking and Applications,Volume:01, Issue:06, Pages: 400-404 (2010).
[14] Ghalem Belalem, Samah Bouamama, Larbi Sekhri, “An Effective Economic Management of Resources in Cloud Computing”, Journal of Computers, Vol. 6, No. 3, March 2011.
[15] Jason C. Hung, Hsing-I Wang, “ Foreword of Special Issue on “E-Service and Applications”, Journal of Computers, Vol. 6, No. 3, March 2011.
[16] Jiehui Ju, Jiyi Wu, Jianqing Fu, Zhijie Lin, Jianlin Zhang, “A Survey on Cloud Storage”, Journal of Computers, Vol. 6, No. 8, August 2011.



Lijuan Zhou received the BE degree in Computer Application Technology from the Heilongjiang University in 1991, the ME degree in Computer Application Technology from the Harbin University of Science And Technology in 1998 and the PhD degree in Computer Application Technology from the Harbin Engineering University in 2004.

She is a professor of database system and data mining at the Capital Normal University. She has conducted research in the areas of database systems, data mining, data warehousing, Web mining, object-oriented database systems, and artificial intelligence, with more than 30 journal or conference publications. Her primary research interests are in OLAP, data mining, and data warehouse.

Xiang Wang received the Bachelor of Engineering degree in Department of Computer Science from North China Institute of Aerospace Engineering in 2010 and he is currently studying for a master's degree at the Capital Normal University. His mentor is Professor Lijuan Zhou and his main research fields are data mining and cloud computing.