Improved Parallel Data Mining Policy for Cloud Computing Environments

Lili Yu[a], Jinzhen Ping[b], Qian Wang[c], Weifeng Wang[d]

Department of Software Engineering, Shijiazhuang Information Engineering Vocational College, Shijiazhuang, 050035, China

[a]76192918@qq.com, [b] 41472796@qq.com, [c]25734476@qq.com, [d]88607232@qq.com

Keywords: cloud computing; data mining; Apriori algorithm; itemset.

Abstract. Cloud computing is a business model. It distributes computing tasks in a large number of computer resource pool configuration. It can provide on-demand for the user computing power, storage capacity and application services capabilities. Cloud computing offers a cheap and efficient solution for storing and analyzing massive amounts of data. Data mining is going to extract useful information and knowledge from a lot of, incomplete, noisy, fuzzy, random data to hidden practice in which people do not know in advance, but is potentially. It has played a guiding role in many fields of scientific research and business decisions ,with far-reaching social and economic significance. Data mining policy for cloud computing environments has important theoretical significance and application value. In this paper, after a series of studies in the improvement of parallel data mining algorithms can greatly improve the efficiency of data mining algorithms.

Introduction

Cloud computing[1] is an emerging method of shared infrastructure, it is based on Internet-centric, open standards and service, it provides a safe, fast and convenient data storage and network computing services. Cloud computing applications to data mining can provide more and more solutions for massive data mining. Cloud computing is dealing with TB or even PB level mass data[2], how to obtain valid information from the data will be the key of cloud computing applications. New ideas, methods and algorithms to accomplish more accurate, fast and powerful data mining are needed in addition to obtain the advantage of parallel computing technology to accelerate the speed of data processing

Google's MapReduce library divided average input files into N pieces, which are uniformly allocated to M nodes[3]. Since the cluster system in the cloud computing environment is heterogeneous, this method does not take full advantage of cluster computing resources. This paper relates to allocation method dataset partitioning, data collection, parallel data mining algorithms based on MapReduce. It introduces the concepts and techniques of cloud computing and data mining, partitioning parallel data mining of existing data sets, parallelism and concurrency strategy, clustering algorithm existing parallel association rule mining algorithms, parallel and parallel classification algorithm[4][5]. Based on this , this paper propses two improved Apriori algorithms to parallelize frequent item sets mining of the huge amounts of data in a cloud computing environment.The improved efficiency of the algorithm can be greatly improved and provide a more effective guarantee for data mining.

Apriori Algorithm Description and Analysis

Algorithm Description. Apriori algorithm is required for frequent item sets mining Boolean association rules algorithm, but also a very influential association rule mining algorithm. Apriori algorithm is based on a priori knowledge about the nature of the frequent item sets named. The algorithm uses an iterative method called layer-by-layer search, generates k+1set from k set. Specifically contains two main processing steps, the connection and delete, take L_k generating from L_{k-1} for example, the mining process are as follows:

(1) Steps to connect: In order to find L_k, use the connection of L_{k-1} with its own set of K candidate set C_k . I_1 and I_2 is assumed in L_{k-1} sets of the two entries, $I_i[j]$ indicates the jth item in I_i. For convenience, Apriori assumes that transactions or item set sort order according to the dictionary entries. For the (k +1) item sets Ii, it means the item sorting, so $I_i[1] < I_i[2] <$..... $I_i[k-1]$. L_{k-1} is denoted by the connecting operation of $L_{k-1} \oplus L_{k-1}$, if the first (k -2) term in I_1 and I_2 is the same, the content in L_{k-1} of I_1 and I_2 can be connected together.

(2) Steps to delete: C_k is a superset of L_k which is not all of the frequent item sets. But all the frequent item sets are certainly in C_k, that means $L_k \subseteq C_k$. Scanning the database can decide support counts of candidate item sets in C_k ,and access to each element in the L_k (frequent item sets). All support count is not less than the minimum support count of candidate frequent item sets which is belong to L_k. However due to the many candidate itemsets in the C_k, this operation is a very large amount of calculation involved.

In order to improve the efficiency of frequent item sets generated one by one, Apriori algorithm is one for narrowing the searching space of frequent item sets of nature: "the loophole all the set of frequent item sets must also be frequent". In other words, if a candidate item sets item k concentration as a subset does not belong to L_{k-1}, then the candidate item sets k and can't be frequent, and thus can be expunged from C_k.

Apriori algorithm has the advantage of simple structure, easy to understand, there are no complicated derivation. In addition, the nature of the algorithm is applied in many cases that have significantly reduced the size of the need to check the candidate, make the algorithm efficiency increased significantly.

The Analysis of Existing Problems.(1) Scanning the database many times: Apriori algorithm in each iteration needs to scan the database once, usually when the length of the excavated maximum frequent item sets is N, it will scan the database for N times. In practical application, we often need to mining long patterns, several times of scanning the database will bring huge overhead.(2) May produce a large number of candidate frequent item sets: Apriori algorithm in the iteration process to produce, processing and preservation in memory candidate frequent item sets, sometimes this quantity is very great. For example: if there are 104 frequent item sets called l, Apriori algorithm will generate nearly 5 x 107 candidate item sets. In addition, in order to find frequent patterns of length 100, it must produce as many as $2^{100} \approx 10^{30}$ candidate, this will result in algorithm on the breadth and depth of adaptability is poor.

Anyhow, Apriori algorithm is based on database of scanning to find all the frequent item sets, the database for storing massive amounts of data more trips to scan will cost a lot of time and memory space, this will become the bottleneck of Apriori algorithm. Therefore, in recent years, studies of parallel data mining algorithm are heating up

Improvement Program

The First Improvement Solutions. Specific improvement ideas are:(1) Divided the transaction database level evenly into size n data subset, a subset of the data sent to m nodes(2) Each node scanning data subset, it generates C_k^p localized collection of candidate item sets k, remember as C_k^p, each candidate item sets k support count is l.(3) Using the partition function m nodes generate the intermediate results of Cp into r different partitions, and then sent to r along with their support count a node.(4) R nodes in the same item sets counting the sum up, to produce the final actual support, compared with the minimum support count min_sup, determine the collection of local frequent item sets k.(5) Combine the output of the node r on generating global frequent item sets collection of L_k.

The advantage of improved Apriori algorithm is to find L_k and L_{k+1} process is completely independent, there is no link between them, it's a cycle process. The k value increasing from 1 to find all frequent item sets. We can also set the k value, you just need to find L_k or from L_1 to L_k between all frequent item sets.

According to the above ideas on Aprior algorithm improved after, we can use the MapReduce programming model as follows:(1) The MapReduce library will be level transaction the database,

divided into n size data subset, the n data subset sent to m a node Map task.(2) Subset of the n data formatting to produce <key_1,$value_1$>, then specific format for <Tid， list>, Here in Tid transaction in the database transaction identifier, list for the transaction in the database transaction value corresponds to the list.(3) The Map function's task is to input data subset of each record<Tid，list> are scanned, produce a local candidate item sets collection of K, denoted by C_k^p. Each candidate item sets support count is 1. The Map function to generate and output in the middle of the<key_1,$value_1$>, here which is defined as < $itemsets_k$, 1>, $itemsets_k$ are used to represent C_k^p of candidate item sets k. Below is the pseudo code of map:

```
map(Tid,list )
//Tid:The transaction identifie
//list:Transaction corresponds to the list
for each k itemsetsk in list:
EmitIntermediate(itemsetsk,1);
```

(4) When a transaction database is very large, after dividing each data subset similar affairs corresponding list, the Map function produced by the intermediate value of key_2 will duplicate data accounts for large proportion. For example, each Map function to produce tens of thousands of such records<{I_1},1>, all of these records will be sent through the network to the designated the Reduce function, it certainly took valuable network resources, increase the delay, reduces the I/O performance. So in each Map tasks on the machine add an optional Combiner function, Combiner function in the first place in the local will be a merger output Map function <$itemsets_k$,sup_k>, sup_k for $itemsets_k$ support in the data subset counting, then using the partition function as shown in formula 1-1, divide the Combiner function produced by the intermediate key/value pair into different r partition, assigned each partition to the specified Reduce function.

$$\text{Hash(key)modR=h}(m_{1,}m_{2,.....}m_k)=(\sum_{j=1}^{k}10^{k-j}m_j\ )\text{mod(R)} \quad (1)$$

These parameters $m_{1,}m_{2,.....}m_k$ means k items in the transaction database entries of centralization of the ordinal numbers, according to Apriori assumption here, transaction or items of concentration is where the sorted sequence according to the dictionary. Below is the pseudo code of Combiner:

```
Combiner(itemsetsk,list(1 ))
// itemsetsk: k item sets
// list(1 ): 1 list
int supk=0;
for each 1 in list:
supk+ =1;
EmitIntermediate(itemsetsk, supk);
```

(5) The Reduce task assigned work site read Combiner function data and submit <$itemsets_k$,sup_k>, due to many different candidate rally k items mapped to the same Reduce function, so prioritize made with the same key $itemsets_k$ candidate item sets k of data aggregation, form <$itemsets_k$, list(sup_k)>.The work site traverse the middle of the data after sorting, pass <$itemsets_k$,list(sup_k)> to Reduce function, then the Reduce function added the same candidate item sets $kitemsets_k$ together to support. Get this candidate item sets k actually support counts in the whole transaction database, and then compared with the minimum support count min_sup, determine the collection of local frequent item sets. Below is the pseudo code of Reduce:

```
Reduce(itemsetsk,list(supk))
// itemsetsk: k item sets
// list(supk): list of count
int result=0;
for each supk
in list:
result+=ParseInt(supk);
```

Emit(itemsetsk, result);

(6) After the comparison of r a Reduce function output of item set and get the final set of frequent item sets k, remember for L_k

(7) That increasing k value from the value of 1 to generate the Lk is empty, so far, has been found all frequent itemsets.

The improved Apriori algorithm implementation by using MapReduce has the advantage as follows: First, the transaction database scanning is less than before; second, the process of finding frequent item sets can be executed in parallel, when finding frequent item sets k process execution of the intermediate results are delivered to the distribution of the Reduce tasks of work site, the Master can dispatching. The Map tasks in the work site began to find frequent k + 1 item sets, speed up the parallel processing, greatly improve the execution efficiency.

The Second Improvement Solutions. The improved algorithm while execution efficiency is higher, but still need to repeatedly scans transaction database in order to get frequent item sets l to frequent item sets k need to scan the transaction database for k times, it will spent a lot of time. So the Apriori algorithm needs to improve again. The algorithm only needs to scan the transaction database again, we can generate all frequent item sets which is still used in cloud computing platform. Specific improvement ideas are as follows:(1) The transaction database level evenly divided into size n data subsets, Subsets of the data sent to m nodes.(2) Each node scan its own data subset, produce a candidate item sets (candidate item sets1 to candidate item sets k) as C^p each support counts of candidate itemsets to 1.(3) R nodes in the same item sets counting the sum up, finally the actual support， compared with the minimum support count min_sup finally determine local frequent itemsets collection of Lp.(4) Combine the output of the node r that produce global frequent itemsets collection of L.

The advantage of improved Apriori algorithm is only need to scan the transaction database again to find all frequent itemsets. According to the above mentality of Apriori algorithm, we can also use the MapReduce programming model.(1) Divided the MapReduce library into n size transactional database subsets, sent the n data subsets to m nodes Map task.(2) Subset of the n data format will produce < key_1, $value_1$ > , Specific format for the <Tid, list>, here Tid means transaction database transaction identifier, list for the transaction in the database transaction value corresponds to the list.(3) The Map function's task is scanning the input data subset of each record <Tid, list> and producing a local collection of candidate item sets, denoted by C^p. Each support counts of candidate item sets to 1, among the Map function to generate and output < key2, value2 >， here defined as the < itemsets, 1 >, itemsets means candidate item sets in C^p. Below is the pseudo code of Map:

```
The map (Tid, list)
/ / Tid: transaction identifier
/ / the list: transaction corresponds to the list
Set itemsets for each item in the list:
EmitIntermediate (itemsets, 1);
```

(4) As these reasons, add an optional Combiner function on each Map tasks machine. First, Combiner function will output the Map function on a combined output < itemsets, sup > to the local place. Sup means item sets in data subset of support counting. Then using the partition function Hash (key) mod R in the Combiner function produced by the intermediate key/value pair into R different partition, Each partition is assigned to the specified the Reduce function. Below is the pseudo code of Combiner:

```
Combiner (itemsets, list (1))
/ / itemsets: itemsets
/ / the list (1) : 1 list
int sup =0;
for each 1 in list:
sup+ =1;
EmitIntermediate(itemsets , sup);
```

(5) Assigned the Reduce task node reads the submitted Combiner function data < itemsets, sup >, because of many different mapping to the same candidate for assembly the Reduce function, making sorted key item sets with the same candidate itemsets data together to form <itemsets,list(sup)> is necessary. Data on the middle of the work site after traversal sequence passed<itemsets,list(sup)> to the Reduce function. Then the Reduce function added the same candidate item sets together to support count, get this candidate item sets in the actual support count, the whole transaction database and then compared with the minimum support count min_sup, determine the collection of local frequent item sets, denoted by L^p. Below is the pseudo code of Reduce:

```
Reduce(itemsets,list(sup ))
/ / itemsets: itemsets
/ / the list (sup) : count list
int result=0;
for each sup in list:
result+=ParseInt(sup);
Emit(itemsets, result);
```

(6) After the comparison of r a Reduce function output of item set and get the final set of frequent itemsets, denoted by L.

The advantage of improved Apriori algorithm using MapReduce implementation is that we can get a complete L frequent item sets only by scanning a transaction databases.

Summary

Efficient data mining is expected by everyone, but when the object of data mining is a huge data set or many of the widely distributed data sources, the efficiency becomes the bottleneck of data mining. With the rapid development of parallel processing technology, a growing number of scientific and technological personnel try to use parallel processing methods to improve the efficiency of data mining. Incorporating the parallel data mining in cloud computing environment, improved the original algorithm of disadvantages will play a huge boost with the huge amounts of data mining technology in cloud computing environment.

References

[1] A Weiss. Computing in Clouds, ACM Networker, 11(4):18-25, Dec.2007.

[2] R Buyya, CS Yeo, S Venugopal, Market-Oriented Cloud Computing:Vision, Hype, and Reality for Delivering IT Services as Computing Utilities. Proceedings of the 2008 10th IEEE International Conference on High Performance Computing and Communications. Vol.00, pp, 5-13, 2008.

[3] D. Gillick, A. Faria, and J. Denero, MapReduce: Distributed Computing for Machine Learning, http://www.icsi.berkeley.edu/~arlo/publications/gillick_cs262a_proj.pdf, 2006.

[4] T. Liu, C. Rosenberg, and H. A. Rowley, Clustering billions of images with large scale nearest neighbor search, in WACV '07: IEEE Workshop on Applications of Computer Vision, 2007.

[5] C. Moretti, K. Steinhaeuser, D. Thain, and N. V. Chawla, Scaling up classifiers to cloud computers, in ICDM'08: Proceedings of the 8th IEEE International Conference on Data Mining. 2008, pp. 472-481.