Implementation of Apriori Algorithm Based on Hadoop Clusters

Kim TongGuk*, Pak CholRyong, Ryang KwangJin
Faculty of Information Science, Kim Il Sung University, Pyongyang, Democratic People’s Republic of Korea
Corresponding Author Email Address: kdg198931@star-co.net.kp

Abstract

With manufacturing technology developing persistently, hardware manufacturing cost becomes lower and lower. More and more computers equipped with multiple CPUs and enormous data disk emerge. Existing programming modes make people unable to make effective use of growing computational resources. Hence cloud computing appears. With the utilization of MapReduce parallelized model, existing computing and storage capabilities are effectively integrated and powerful distributed computing ability is provided.

Association rules can forcefully get a horizontal relation in the big data, the Apriori algorithm is one of the most significant association rules. Traditional mining based on parallel Apriori algorithms needs much more time in data IO with the increasing size of large transaction databases. This paper improves the Apriori algorithm from compressing transactions, reducing the number of scans and simplifying candidate set generation. And then the improved algorithm is parallelized on the Hadoop framework. The experiments show that this improved algorithm is suitable for large-scale data mining and has good scalability and effectiveness.

Keywords: Apriori algorithm, Association rules, Data Mining, MapReduce, Hadoop

1. Introduction

In the context of the development of big data “spraying wells”, there is frequently a close relationship between vast amount of data [1]. Analysis and decision making through data mining have become the mainstream of social development. In order to better find the relevance of transaction datasets, some researchers have discovered the concept of association rule mining technology [2]. With the attention of many researchers at home and abroad caused by the conception of the association rule mining, they have done a lot of analysis in this field and put forward a lot of data mining algorithms.

One of the most famous association rule algorithms is the Apriori algorithm, which is a classic association rule algorithm designed by Agrawal [3-4] in 1994. It is a level-by-level search iteration method that constructs a k-item set to constitute a k+1-item set. The main ideas of this algorithm are: Firstly, all frequency sets are counted from the transaction database, and the support of this frequent set must not be less than the minimum support degree; Secondly it enters into the process of strong association rule generation, and the rules need to satisfy the support and confidence thresholds at the same time; Thirdly, only all rules that contain collection items are retained. Once these rules are retained and generated, that are greater than or equal to the MinConfidence.

Due to emergence of cloud computing, it’s possible to get big cheap computing and storage ability quickly and dynamically, solving the most fundamental problem for data
mining about how to acquire inexpensively powerful data computing ability [5-8]. Related researchers directed attentions to cloud computing platform, in the hope of implementing data mining algorithm with high scalability, applying cheap computing of cloud computing to data mining based on storage ability, thus overcoming the shortcomings in traditional data mining, reducing calculation cost and enhancing data mining efficiency [9-14].

With a view on the broad and promising future of cloud computing, the integration of studying and applying cloud computing and existing data mining algorithm has become hot concern in various industries [15-18].

The design of the Hadoop [19] framework originated was from an open source project developed by the Apache organization Foundation. Because of its inter-temporal significance, the Hadoop framework has been widely used in the information field at home and abroad. There are two important modules in the Hadoop frame—Distributed File System HDFS and Distributed Computing Frame MapReduce [20]. As a distributed file system, HDFS functions aim to implement data storage. It will work in conjunction with the computational framework. MapReduce works to provide the underlying support for data calculations. And the idea of MapReduce [21] is based on a paper by Google. In short, its core method is “the decomposition of tasks and the statute of results.”

Here we transform classical data mining correlative algorithm Apriori into implementing in cloud computing environment based on MapReduce model; meanwhile, according to characteristics of MapReduce model, we improve Apriorialgorithm to make MapReduce-Apriorialgorithm with strong scalability, fit for tremendous data analysis and processing. Finally, utilize Map Reduce-Apriori parallel algorithm to test the proposed method with running time from the perspective of data volume and computing node quantity to get practically meaningful data mining results [18].

2. Brief and Research Status of Apriori Algorithm

2.1 Overview of Apriori Algorithm

The Apriori algorithm is an iterative level-by-level search method that consists of a k-item set to construct a (k+1)-item set. First, obtain a frequent 1-item set. Then, obtain a frequent 1-item set.

L1 can generate a frequent 2-item set L2, and L2 can generate a frequent 3-item set L3. According to this rule, when a frequent k-item set cannot be found, the algorithm ends [22]. The specific operation is as follows:

1) Iterate through the initial transaction database and count the frequency of occurrence of the candidate set. The result is the support of the project. All projects whose all supports level no lower than the preset threshold generate a frequent 1-item set L1.
2) The algorithm uses L1 JOIN L1 to form a candidate C2-item set C2.
3) Using the items in C2, traverse the database again to obtain the support degree of each candidate set. All projects with support levels not lower than the support level generate frequent 2-item set L2.
4) The algorithm uses L2 JOIN L2 to form a candidate 3-item set C3 of candidate 3-item sets.
5) Using the items in C3 to traverse the database again, the support degree of each candidate set can be obtained. All items with support levels not lower than the support level generate frequent 3-item set L3.

The above process is performed iteratively until the candidate set Ck is empty. The Apriori algorithm does multiple IO operations on the database. Each stage consists of two parts, namely connection and pruning.

2.2 The shortcomings of Apriori algorithm
1) When the Apriori algorithm generates the candidate item set, it needs to perform the self-connection operation on the frequent item sets obtained in the previous step. Then scan the transaction data set again and compare the candidate set formed by the self-connection with min_sup. During the self-connection operation, a large amount of comparison work will be performed.

2) Apriori algorithm needs to rescan transaction datasets before pruning, and then compare with min_sup. Therefore, when the size of the transaction dataset is getting larger and larger, each scan will consume a lot of time, resulting in inefficient mining.

3) In the current situation where the data information has a high dimension and the type is complex, the classical Apriori algorithm can't satisfy users.

4) Because the classic Apriori algorithm is only applicable to a single machine, when the size of transaction datasets gradually becomes larger and larger, it will lead to inefficient mining, insufficient storage space, and even system crashes.

3. Implementation of Apriori Algorithm Based on Hadoop Clusters

3.1 Reduce frequent item sets self-connection comparison times and pruning steps

In this paper, a method to reduce data when scan for candidate set has been introduced.

If n-dimensional data set is not considered as frequent set, then its n-1 dimensional data set is also not the frequent data set so that by comparing and deleting not-frequent data set is finally resulted in smaller size of data to scan leading to high efficiency of scanning algorithm.

3.2 Reduce the Number of Scanned Databases

It is kind of problematic to scan database for scanning frequent item sets because of frequent I/O but making the database as vertical data table effectively reduce the number of scanned databases because finally it is resulted in scanning one transaction database.

3.3 Combining Apriori Algorithm and Hadoop Platform

3.3.1 Data Initialization

The symbols used in this paper are described in table 1.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$L_k$</td>
<td>Frequent sets of the kth layer</td>
</tr>
<tr>
<td>GMap</td>
<td>First layer frequently set the item set to id mapping table</td>
</tr>
<tr>
<td>$LM_k$</td>
<td>Ordered and mapped to the kth layer of id in GMap frequent sets</td>
</tr>
<tr>
<td>$C_k$</td>
<td>Candidate sets for the kth layer</td>
</tr>
<tr>
<td>$S_k$</td>
<td>A superset of the kth layer</td>
</tr>
<tr>
<td>MP</td>
<td>Master mode</td>
</tr>
<tr>
<td>$MPS_k$</td>
<td>Master mode (k-1) sub mode</td>
</tr>
<tr>
<td>GP</td>
<td>Generation model</td>
</tr>
<tr>
<td>GPS</td>
<td>Generation mode base</td>
</tr>
</tbody>
</table>
After producing first frequent itemset, then arrange it and combine it in frequent set so that simpler calculation is implemented without complex comparisons, still, it is needed to do huge amount of calculation so that it can be considered as independent stage called data initializing stage.

Three steps have been done through data initializing stage.

1) Producing frequent itemsets
2) Ordering frequent itemsets
3) Generating second candidate itemsets.

The below pseudo code is parallelized process which is for data initializing stage to generate second frequent itemsets $C_2$ basing on Map Reduce model. Here $C_2$ is ordered.

```
//MapReduce Stage1-1
Mapper{
    Map() {
        For each itemsets in value
        Key=itemsets
        Value=1
        Emit(key,value)
    }
}

//Stage1-2
LM1=SortOutputAndMap(L1)

//Stage1-3
Mapper{
    Map() {
        For i=0; i< LM1.size()-1; i++
        For j=i+1; j< LM1.size()-1; j++
        Emit(LM1[i], LM1[j])
    }
}
```

Through the second stage, results from the first stages are sorted by support degree and then ranked into digital id which is called GMap. GMap makes the data easier to treat because it transforms the character expressed data into numerical group attributing to high efficiency of data calculation such as comparison.

The results of second stage is second candidate itemsets which is grouped by host nodes. Table 2 shows the dataflow thru the entire data initializing stage.

<table>
<thead>
<tr>
<th>Original input</th>
<th>After sorting the frequent sets (minimum support degree is 50%)</th>
<th>Calculate the 1st itemset</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T$</td>
<td>$T_k$</td>
<td></td>
</tr>
<tr>
<td>Original transaction set</td>
<td>kth layer transaction set after compression</td>
<td></td>
</tr>
</tbody>
</table>
3.3.2 Iterative implementation

Finding frequent itemsets could be carried out by iterating bellow two operations. These operations are basing on
dataset from data initializing stage.

1) Compute the kth frequent itemset

In this stage all data is loaded to internal memory so that increase the efficiency of computation. Input data to Map
is files’ row data and the output key is value of each column. The pseudo-codes are implemented as follows:

```
//MapReduce Stage2-1
 Mapper{
     Setup()
     Map() {
         Value=MapToid(value, GMap)
         Value.sort()
         For each itemsets in
         If(value.contains(itemsets))
         Emit(itemsets, 1)
     }
 }
 Reducer {
     Reduce() {
         sum=0
         For each value in values
         sum=sum + 1
         If sum > minsup
         Emit(key, sum)
     }
 }
```

Table 3. Calculating the Data of 2nd Layer Frequent Sets

<table>
<thead>
<tr>
<th>Original input</th>
<th>Map stages</th>
<th>Reduce stages</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mapping and scheduling</td>
<td>output</td>
</tr>
<tr>
<td>acfg</td>
<td>1,2</td>
<td>[1,2],1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[1,2],1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[1,2],1</td>
</tr>
<tr>
<td>abcde</td>
<td>1,2,3</td>
<td>[1,2],1</td>
</tr>
<tr>
<td>bdgac</td>
<td></td>
<td>[1,3],1</td>
</tr>
<tr>
<td>adc</td>
<td></td>
<td>[1,3],1</td>
</tr>
<tr>
<td>fgh</td>
<td></td>
<td>[1,3],1</td>
</tr>
</tbody>
</table>
2) Compute the (k+1)th candidate itemset

This stage includes two steps of implementation:

(1) produce superset at (k+1)th layer from the kth frequent sets;
(2) trim superset at (k+1)th layer to generate candidate itemsets at (k+1)th layer. So to carryout, this stage can be divided into two stages: Map and Reduce stage. Figure 1 shows the data iteration phase.

![Figure 1. The Third Layer Data Generated Superset Node Distribution](image)

4. Generation of Association Rules

Association rules are generated when the supporting rate is bigger than the minimum confidence degree after calculation of the rate in frequent itemsets. It’s done in following steps: for given frequent itemsets I producing association rules, check 1 each non-empty subset to get a relative rule \( a \Rightarrow (1 - a) \) – and its confidence degree is \( \text{support}(I) / \text{support}(a) \); when confidence degree is bigger than the minimum confidence, the association rule is produced.

In this process, one thing to consider is that if the association rule produced by the maximum subset of frequent itemset do not meet the requirement of minimum confidence, then it is obvious that any subset frequent itemsets also cannot meet the minimum confidence requirement as well.

Take for instance frequent itemset \([1, 2, 3, 4]\). If the confidence degree of \(1, 2, 3 \Rightarrow 4\) can’t suffice the minimum value, it’s inevitably the confidence of \(1, 2 \Rightarrow 3, 4\) can’t suffice the minimum degree, without consideration of subset. Hence by that feature, we can improve efficiency of overall operation during actual computation. To parallelize the process of producing association rules, we can assign each infrequent itemset to different Map for generating simultaneously. So the parallel generation of association rules based on the MapReduce model pseudo code as follows:

```java
// MapReduce Stage 3
Mapper{
    Map() {
        a=I-1
    }
```
i=1
While(confidence(l,a)>minisupport&&i+1{
    i=i++
    Emit(\(a \Rightarrow (1 - a)\) , confidence(l,a)
    a=l-1 })
Reducer{
    Reduce()
    Emit(key,value);
}};

5. Experimental Analysis and Results

This section evaluates the performance of the proposed algorithm by comparing its execution time performance against proposed algorithm.

To analyze the proposed algorithm, we set up a cluster of 3 nodes. They have Intel(R) Core (TM) i3-4500U CPU @ 1.80GHz, 2401 MHz, 2 Core(s), 4 Logical Processor(s). All nodes have 8GB RAM, and a 500GB hard disk. The worker nodes are installed on Ubuntu 18.04, Hadoop 2.7.1. One is for namenode and two of others are working as datanode. Replication is set to 2 and block size is 128MB.

Experiment 1: Performance Comparison between Apriori Algorithm and Proposed Apriori Algorithm

The transaction data set for this experiment is stored as a file. Performance analysis of mining time before and after improved with 3 nodes Hadoop cluster test algorithm. First, on the premise that the number of nodes in the Hadoop cluster is unchanged, continuously increase the number of itemsets in the experimental data itemset, and set the minimum support to the same, that is, min_sup=0.3. The experimental results are shown in Table 4.

<table>
<thead>
<tr>
<th>Trasaction itemsets</th>
<th>Apriori Algorithm(s)</th>
<th>Proposed Algorithm(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1500</td>
<td>13.6</td>
<td>7.3</td>
</tr>
<tr>
<td>3000</td>
<td>19.1</td>
<td>10.6</td>
</tr>
<tr>
<td>4500</td>
<td>28.6</td>
<td>16.2</td>
</tr>
<tr>
<td>6000</td>
<td>40.2</td>
<td>28.9</td>
</tr>
<tr>
<td>7500</td>
<td>60.4</td>
<td>42.8</td>
</tr>
</tbody>
</table>

From the Table 4, proposed algorithm is often better than classical Apriori algorithm in temporal performance, and with the increasing number of transaction item sets, apriori algorithm running on a computer can significantly improve the time of mining analysis. However, with the proposed algorithm, as the number of transaction item sets increases, the time performance is getting better and better. Because with the increase in the number of transaction items, the nodes of the distributed cluster will gradually increase. In summary, the improved proposed algorithm is superior to the apriori algorithm in temporal performance.

Experiment 2: Performance Comparison between Apriori Algorithm and Proposed Algorithm under Different Supporting Degrees.
First, this paper tests the data set RETAIL, selects the minimum support threshold range [0.02, 0.20]. And within this range, evenly increase the step: 0.02, so there will be a threshold of 10. Then, this paper uses the data set retail to run the Apriori algorithm and the proposed algorithm respectively, and record the running time (Note that the running time is second). Figure 2 shows the experimental data obtained by executing the above three algorithms. Horizontal axis: support; vertical axis: time/s.

Experiments show that the proposed algorithm runs much less time than the Apriori algorithm under different support levels. The higher the support, the longer the Apriori algorithm will run than the proposed algorithm. In summary, the temporal performance of the proposed algorithm under different support levels is always superior to the traditional Apriori algorithm.

![Figure 2. Performance Comparison under different support levels](image)

6. Conclusion

Aiming at the traditional Apriori algorithm, when mining frequent itemsets, you need to continuously scan transaction data sets, so that the system I/O overhead and other shortcomings. In this paper, we improved Apriori algorithm in three aspects: compression in the transaction, reducing the number of scanning areas, and simplifying the candidate set generation. At the same time, the improved algorithm is parallelized in the Hadoop framework. The simulation results show that compared with the traditional Apriori algorithm, the proposed algorithm has good performance and security in temporal performance, mining frequent candidate itemsets and different support levels. However, it needs to be continuously improved in the future work.

References


