Apriori-Map/Reduce Algorithm

Jongwook Woo
Computer Information Systems Department
California State University
Los Angeles, CA

Abstract—Map/Reduce algorithm has received highlights as cloud computing services with Hadoop frameworks were provided. Thus, there have been many approaches to convert many sequential algorithms to the corresponding Map/Reduce algorithms. The paper presents Map/Reduce algorithm of the legacy Apriori algorithm that has been popular to collect the item sets frequently occurred in order to compose Association Rule in Data Mining. Theoretically, it shows that the proposed algorithm provides high performance computing depending on the number of Map and Reduce nodes.

Keywords: Map/Reduce, apriori algorithm, Data Mining, Association Rule, Hadoop, Cloud Computing

1 Introduction

People started looking at and implementing Map/Reduce algorithm for most of applications, especially for computing Big Data that are greater than peta-bytes as cloud computing services are provided, for example, by Amazon AWS.

Big Data has been generated in the areas of business application such as smart phone and social networking applications. Especially these days, the better computing power is more necessary in the area of Data Mining, which analyzes tera- or peta-bytes of data. Thus, the paper presents Apriori-Map/Reduce Algorithm that implements and executes Apriori algorithm on Map/Reduce framework.

In this paper, section 2 is related work. Section 3 describes the legacy apriori algorithm. Section 4 introduces Map/Reduce and Hadoop and presents the proposed Apriori-Map/Reduce algorithm. Section 5 is conclusion.

2 Related Work

Association Rule or Affinity Analysis is the fundamental data mining analysis to find the co-occurrence relationships like purchase behavior of customers. The analysis is legacy in sequential computation so that many data mining books never resist illustrating it.

Aster Data has SQL MapReduce framework as a product [9]. Aster provides nPath SQL to process big data stored in the DB. Market Basket Analysis is executed on the framework but it is based on its SQL API with MapReduce Database.

Woo et al [11-13] presents Market Basket Analysis algorithms with Map/Reduce, which proposes the algorithm with (key, value) pair and execute the code on Map/Reduce platform. However, it does not use the apriori property but instead adopts joining function to produce paired items, which possibly computes unnecessary data.

3 Apriori Algorithm

Apriori algorithm shown in Figure 3.1 has been used to generate the frequent item sets in the amount of data transactions in order to produce an association rule.

![Figure 3.1. Apriori Algorithm](image)

The association rule has been used efficiently to manage stock items and products etc analyzing the customer’s behavior. It is based on Apriority Property where all subsets of a frequent item set must also be frequent.

For example, minimum support is 2 and there are size 2 item sets generated: `<coffee, cracker>, 3>` and `<coke, cracker, cracker>`. 
cracker], 1> as <[item pairs], frequency>. And, when there are size 3 item sets produced as <[coffee, cracker, milk]> and <[coke, cracker, milk]>, as Apriori Property, <[coke, cracker, milk]> is eliminated before counting the frequencies of the item sets in the transaction data, which reduce unnecessary computing time.

The time complexity of the algorithm is \(O(k x (k^3 + t x n))\) when \(k\): size of frequent items, \(t\): transaction data, \(n\): number of item elements in each transaction \(t\). It is simplified to \(O(k^3 + k x t x n)\) and then \(O(k x t x n)\) where \(t >> k, n>>k\).

4 Apriori-Map/Reduce Algorithm

4.1 Map/Reduce in Hadoop

Map/Reduce is an algorithm used in Artificial Intelligence as functional programming. It has been received the highlight since re-introduced by Google to solve the problems to analyze Big Data, defined as more than petabytes of data in distributed computing environment. It is composed of two functions to specify, “Map” and “Reduce”. They are both defined to process data structured in (key, value) pairs.

Inspired by Google's MapReduce and GFS (Google File Systems) [1], Map/Reduce platform is implemented as Apache Hadoop project that develops open-source software for reliable, scalable, and distributed computing. Hadoop can compose hundreds of nodes that process and compute Big Data. Hadoop has been used by a global community of contributors such as Yahoo, Facebook, Cloudera, and Twitters etc. Hadoop provides many subprojects including Hadoop Common, HDFS, MapReduce, Avro, Chukwa, HBase, Hive, Mahout, Pig, and ZooKeeper etc [2].

The map and reduce functions run on distributed nodes in parallel. Each map and reduce operation can be processed independently on each node and all the operations can be performed in parallel. Map/Reduce can handle Big Data sets as data are distributed on HDFS (Hadoop Distributed File Systems) and operations move close to data for better performance [5].

Hadoop is restricted or partial parallel programming platform because it needs to collect data of (key, value) pairs as input and parallely computes and generates the list of (key, value) as output. In map function, the master node divides the input into smaller sub-problems, and distributes those to worker nodes.

Figure 4.1 illustrates Map/Reduce control flow where, as \(m\) is map node id, each valuemn is simply 1 and gets accumulated for the occurrence of items together, which is clearly illustrated in Market Basket Analysis Algorithm of Woo et al [11-13]. Map function takes inputs \((k_i, v_i)\) and generates \(<k_i, v_i>\) where \(<\) represents list or set. Combiner function that resides on map node takes inputs \((k_2, <v_2>)\) and generates \(<k_2, v_2>\). Reduce function takes inputs \((k_3, <v_3>)\) and generates \(<k_3, v_3>\).
Map/Reduce approach, especially for big data set, gives us the opportunity to develop new systems and evolve IT in parallel computing environment. The approach started a few years ago but many IT companies in the world already have adapted to Map/Reduce approach.

4.2 Apriori-Map/Reduce Algorithm

Figure 4.2 is the proposed Apriori-Map/Reduce Algorithm that runs on parallel Map/Reduce framework such as Apache Hadoop. prune(C_{k+1}) function is to remove the non-frequent item set C_{k+1} by eliminating non-frequent item sets C_{k} as non-frequent item sets cannot be a subset of frequent item sets.

The algorithm starts with (1) that calculates frequent item set for each map node as the time complexity O(t/m x n) where t: number of transactions, n: number of items in the transactions, m: num of map nodes. Then, (2) are to collect the frequent item set and (3) is to remove items that does not meet the minimum support in reduce nodes as O(t/r x n) when r: number of reduce nodes. The time complexity of (1-3) can be simplified initially to O((t/m + t/r) x n) and then O(t x n / x) when m = r = p.

(4) is to calculate frequent item set with an additional item by joining, sorting, and eliminating the duplicated items in each map node where join_sort is O(k x k / m) when k: size of frequent items and prune is O((k-1) x k / r) that is simplified to O(k^2 / r). Similarly, (5) is to collect the frequent item set at the reduce nodes. (6) is to count item frequencies that do not meet the minimum support at the reduce nodes. (7) is to remove items that does not meet the minimum support in reduce nodes as O(t/r x n) that is initially simplified to O((t/m + t/r) x n) and then O(t x n / p) when m = r = p.

4.2.1 Time Complexity

The overall time complexity of Apriori-Map/Reduce Algorithm is calculated as O(k x (k^2 + t x n)/p) where k: size of frequent items, t: number of transactions, n: number of items in the transactions, p: number of map and reduce nodes assuming the node sizes are the same. And, it becomes O((k^3 + k x t x n)/p) and then O(k x t x n/p) where t >> k, n >> k. It theoretically shows that the time complexity is p times less than the sequential apriori algorithm.

4.2.2 Example of the Algorithm

Figure 4.3 is the example transaction data at a store to explain how the proposed algorithm works.

| Transaction 1: cracker, beer |
| Transaction 2: chicken, pizza, |
| Transaction 3: coke, cracker, beer |
| Transaction 4: coke, cracker |
| Transaction 5: beer, chicken, coke |
| Transaction 6: chicken, coke |

Figure 4.3 Transaction data at a store

Suppose that minimum support is 2/6 that represents two items out of 6 transactions as 33%.

(a) The first step

Assuming there are three Map nodes, two transaction data are distributed to three map nodes, that is, map node 1 has transaction data 1 and 2. Node 2 has transaction data 3 and 4. Node 3 has transaction data 5 and 6. Therefore, we can generate size 1 frequent items C_{m1} with an item pair set <item, frequency> at the map node m.

C_{11} = {<cracker, 1>, <beer, 1>, <chicken, 1>, <pizza, 1>}
C_{21} = {<coke, 2>, <cracker, 2>, <beer, 1>}
C_{31} = {<beer, 1>, <chicken, 2>, <coke, 2>}

From all C_{m1}, reduce nodes collect and compute C_{i} that is size 1 frequent item pairs and L_{i} that is size 1 frequent item pairs that meet minimum support. Thus, [pizza, 1] of C_{i} is eliminated in L_{i}.

C_{i} = [{<cracker, 3}, <beer, 3>, <chicken, 3>, [pizza, 1], [coke, 4]}];
L_{i} = [{<cracker, 3}, <beer, 3>, <chicken, 3}, [coke, 4]}

(b) The second step

In the loop, L_{i} is mapped to each map node m for L_{m1}.

L_{11} = {cracker, beer}
L_{21} = {chicken}
L_{31} = {coke}

Then, size 2 frequent item pair sets can be generated by joining and sorting L_{i} to each item sets of the map node m as C_{m2} = L_{j} join_sort L_{m1} where the duplicated item sets are eliminated:

C_{12} = {<beer, cracker>, <chicken, cracker>, <beer, chicken>, <coke, cracker>, <beer, coke>}
C_{22} = {<chicken, cracker>, <beer, chicken>, <chicken, coke>}
C_{32} = {<coke, cracker>, <beer, coke>, <chicken, coke>}

From all C_{m2}, reduce nodes collect C_{2} that is size 2 frequent item pairs.

C_{2} = {<beer, cracker>, <chicken, cracker>, <beer, chicken>, <coke, cracker>, <beer, coke>, <chicken, coke>};

C2 is mapped to each map node as C_{m2} as follows:

C_{12} = {<beer, cracker>, <chicken, cracker>}
C_{22} = {<beer, chicken>, <coke, cracker>}
C_{32} = {<beer, coke>, <chicken, coke>}

Now, we can generate size 2 frequent items with an item pair set $[\text{item, frequency}]$ at the node $m$ that contains all transaction data as presented in Figure 4.2.

$$C_{12} = \{ [<\text{beer, cracker}>, 2], [<\text{chicken, cracker}>, 0] \}$$

$$C_{22} = \{ [<\text{beer, chicken}>, 1], [<\text{coke, cracker}>, 2] \}$$

$$C_{32} = \{ [<\text{beer, coke}>, 2], [<\text{chicken, coke}>, 2] \}$$

From all $C_{m3}$, the reduce nodes collect and compute $C_3$ that is size 2 frequent item pairs and $L_2$ that is size 2 frequent item pairs that meet minimum support. Thus, $[<\text{chicken, cracker}>, 0]$ and $[<\text{beer, chicken}>, 1]$ are eliminated from $C_3$ for $L_2$:

$$C_3 = \{ [<\text{beer, cracker}>, 2], [<\text{chicken, cracker}>, 0], [<\text{beer, chicken}>, 1], [<\text{coke, cracker}>, 2], [<\text{beer, coke}>, 2], [<\text{chicken, coke}>, 2] \}$$

$$L_2 = \{ [<\text{beer, cracker}>, 2], [<\text{coke, cracker}>, 2], [<\text{beer, coke}>, 2], [<\text{chicken, coke}>, 2] \}$$

(c) The third step

In the loop, $L_2$ is mapped to each map node $m$.

$$L_{12} = \{ <\text{beer, cracker}> , <\text{coke, cracker}> \}$$

$$L_{22} = \{ <\text{beer, coke}> \}$$

$$L_{32} = \{ <\text{chicken, coke}> \}$$

Then, size 3 frequent item pair sets can be generated by joining and sorting $L_2$ to each item sets of the map node $m$ as $C_{m3} = L_2 \text{join_sort} L_{m2}$ where the duplicated item sets are eliminated:

$$C_{13} = \{ <\text{beer, coke, cracker}>, <\text{beer, chicken, coke}>, <\text{chicken, coke, cracker}> \}$$

$$C_{23} = \{ <\text{beer, coke, cracker}>, <\text{beer, chicken, coke}> \}$$

$$C_{33} = \{ <\text{beer, chicken, coke}>, <\text{chicken, coke, cracker}> \}$$

As the item size $k$ is or greater than 3, $\text{prune}(C_{m3})$ deletes non frequent item sets that violates apriori property that all subsets of a frequent item set must also be frequent:

$$C_{13} = \{ <\text{beer, coke, cracker}> \}$$

$$C_{23} = \{ <\text{beer, coke, cracker}> \}$$

$$C_{33} = \{ \}$$

$C_{13}$, $C_{23}$, $C_{33}$ eliminate $<\text{beer, chicken, coke}>$ and $C_{13}$, $C_{33}$ removes $<\text{chicken, coke, cracker}>$ as $<\text{beer, chicken}>$ and $<\text{chicken, cracker}>$ respectively are not a member of $L_2$, that is, non-frequent items.

From all $C_{m3}$, reduce nodes collect $C_3$ that is size 3 frequent item pairs and $L_3$ that is size 3 frequent item pairs that meet minimum support:

$$C_3 = \{ <\text{beer, coke, cracker}> \}$$

$$L_3 = \{ \}$$

Since $L_3$ does not have any element, the algorithm ends. Therefore, we have frequent item sets $L_1$ and $L_2$ with size 1 and 2 respectively:

$$L_1 = \{ [\text{cracker}, 3], [\text{beer}, 3], [\text{chicken}, 3], [\text{coke}, 4] \}$$

$$L_2 = \{ [\text{beer, cracker}>, 2], [\text{coke, cracker}>, 2], [\text{beer, coke}>, 2], [\text{chicken, coke}>, 2] \}$$

The item sets $L_1$ and $L_2$ can be used to produce association rule of the transaction.

5 Conclusion

The paper proposes Apriori-Map/Reduce Algorithm and illustrates its time complexity, which theoretically shows that the algorithm gains much higher performance than the sequential algorithm as the map and reduce nodes get added. The item sets produced by the algorithm can be adopted to compute and produce Association Rule for market analysis.

The future work is to build the code following the algorithm on Hadoop frame and generate experimental data by executing the code with the sample transaction data, which practically proves that the proposed algorithm works. Besides, the algorithm should be extended to produce association rule.

6 Reference


