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.

```
Map transaction t in data source to all Map nodes;
//(1)
C1 = {size 1 frequent items};
// (2) min_support = num / total items; for example: 33%
L1 = {size 1 frequent items ∩ min_support};


for (k = 1; Lk !=∅; k++) do begin


    // (3) sort to remove duplicated items
    Ck+1 = Lk join_sort Lk;

    for each transaction t in data source with Ck+1 do
        // (4)
        increment the count of all candidates in Ck+1 that
        are contained in t
        // (5) find Lk+1 with Ck+1 and min_support
        Lk+1 = {size k+1 frequent items ∩
          min_support};
    end
end


return ∪k Lk;
```

Figure 3.1. Apriori Algorithm

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], 1> as *<[item pairs], frequency>*. And, when there are size 3 item sets produced as *<[coffee, cracker, milk]>* and *<[coke, cracker, milk]>,* as *Apriority 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 $O(k \times (k^2 + t \times n))$ when k: size of frequent items, t: transaction data, n: number of item elements in each transaction t. It is simplified to $O(k^3 + k \times t \times n)$ and then $O(k \times t \times n)$ 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 peta-bytes 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].

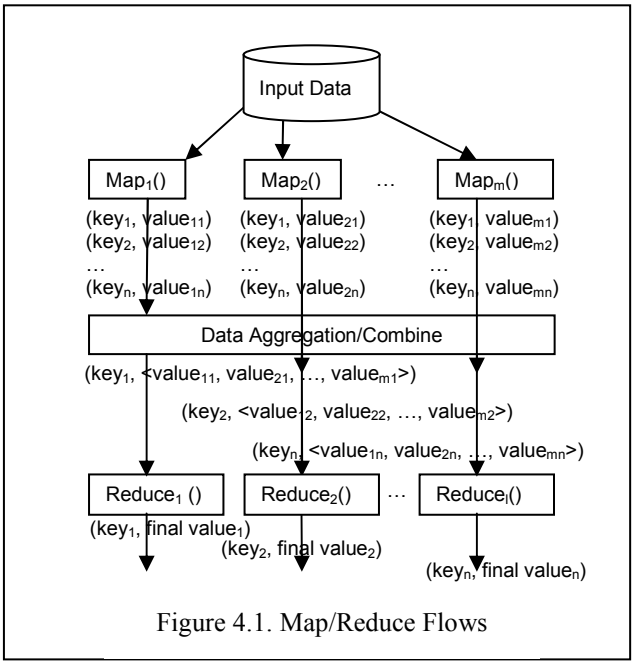

Figure 4.1. Map/Reduce Flows

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.

Map transaction *t* in data source to all Map nodes;
//(1) In each Map node *m*
C_{m1} = {size 1 frequent items at the node *m*};
// (2) In Reduce, compute C_1 and L_1 with all C_{m1} ;
C_1 = {size 1 frequent items};
// (3) min_support = num / total items; for example: 33%
L_1 = {size 1 frequent items ∩ min_support};

for (k = 1; L_k !=∅; k++) do begin

// (4) In each Map node *m*
// L_{mk}: L_k mapped to each node m;
// sort to remove duplicated items
$C_{m(k+1)} = L_k$ *join_sort* L_{mk};

// (5) In Reduce, use Apriori Property
compute C_{k+1} with all sorted $C_{m(k+1)}$;
if (k>=3) prune(C_{k+1});

for each transaction *t* in data source with C_{k+1} do
// (6) In each Map node *m*
increment the count of all candidates in $L_{m(k+1)}$ that are contained in *t*
end
// (7) In Reduce, find L_{k+1} with $L_{m(k+1)}$ and
// min_support
L_{k+1} = {size *k+1* frequent items ∩ min_support};
end

return ∪k L_k;

Figure 4.2. Apriori-Map/Reduce Algorithm

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_1, v_1) and generates $<k_2, v_2>$ 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 $(k_2, <v_2>)$ 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 $O(t/m \times n)$ when 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 $O(t/r \times n)$ when r: number of reduce nodes. The time complexity of (1-3) can be simplified initially to $O((t/m + t/r) \times n)$ and then $O(t \times n / x)$ 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 $O(k \times k / m)$ when k: size of frequent items and *prune* is $O((k-1) \times k / r)$ 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 map nodes as $O(t/m \times n)$. (7) is to remove items that does not meet the minimum support in reduce nodes as $O(t/r \times n)$ that is initially simplified to $O((t/m + t/r) \times n)$ and then $O(t \times n / p)$ when $m = r = p$.

4.2.1 Time Complexity

The overall time complexity of *Apriori-Map/Reduce Algorithm* is calculated as $O(k \times (k2 + t \times n)/p)$ 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 $O((k3 + k \times t \times n)/p)$ and then $O(k \times t \times n/p)$ 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_1 that is size 1 frequent item pairs and L_1 that is size 1 frequent item pairs that meet minimum support. Thus, *[pizza, 1]* of C_1 is eliminated in L_1.

C_1 = {[cracker, 3], [beer, 3], [chicken, 3], [pizza, 1], [coke, 4]};
L_1 = {[cracker, 3], [beer, 3], [chicken, 3], [coke, 4]}

(b) The second step
In the loop, L_1 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_1 to each item sets of the map node m as $C_{m2} = L_1$ *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 [*item, frequency]* at the node *m* that contains all transaction data as presented in Figure 4.2.

C_{12} = {[<beer, cracker>, 2], [<chicken, cracker>, 0]}
C_{22} = {[<beer, chicken>, 1], [<coke, cracker >, 2]}
C_{32} = {[<beer, coke>, 2], [<chicken, coke>, 2]}

From all C_{m2}, the reduce nodes collect and compute C_2 that is size 2 frequent item pairs and L_2 that is size 2 frequent item pairs that meet minimum support. Thus, *[<chicken, cracker >, 0]* and *[<beer, chicken >, 1]* are eliminated from C_2 for L_2:

C_2 = {[<beer, cracker>, 2], [<chicken, cracker >, 0], [<beer, chicken >, 1], [<coke, cracker>, 2], [<beer, coke>, 2], [<chicken, coke>, 2]};
L_2 = {[<beer, cracker>, 2], [<coke, cracker>, 2], [<beer, coke>, 2], [<chicken, coke>, 2]};

(c) The third step
In the loop, L_2 is mapped to each map node *m*.

L_{12} = {<beer, cracker>, <coke, cracker>}
L_{22} = {<beer, coke>}
L_{32} = {<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 $C_{m3} = L_2$ *join_sort* L_{m2} where the duplicated item sets are eliminated:

C_{13} = {<beer, coke, cracker>, <beer, chicken, coke>, <chicken, coke, cracker>}
C_{23} = {<beer, coke, cracker>, <beer, chicken, coke>}
C_{33} = {<beer, chicken, coke>, <chicken, coke, cracker>}

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

C_{13} = {<beer, coke, cracker>}
C_{23} = {<beer, coke, cracker>}
C_{33} = {}

C_{13}, C_{23}, C_{33} eliminate <*beer, chicken, coke*> and C_{13}, C_{33} removes <*chicken, coke, cracker*> as <*beer, chicken*> and <*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 = {<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:

L1 = {[cracker, 3], [beer, 3], [chicken, 3], [coke, 4]}
L2 = {[<beer, cracker>, 2], [<coke, cracker>, 2], [<beer, coke>, 2], [<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

[1] "MapReduce: Simplified Data Processing on Large Clusters", Jeffrey Dean and Sanjay Ghemawa, Google Labs, pp. 137–150, OSDI 2004

[2] Apache Hadoop Project, http://hadoop.apache.org/,

[3] "Building a business on an open source distributed computing", Bradford Stephens , Oreilly Open Source Convention (OSCON) 2009, July 20-24, 2009, San Jose, CA

[4] "MapReduce Debates and Schema-Free", Woohyun Kim, Coord, March 3 2010

[5] "Data-Intensive Text Processing with MapReduce", Jimmy Lin and Chris Dyer, Tutorial at the 11th Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL HLT 2010), June 2010, Los Angeles, California

[6] "SQL MapReduce framework ", Aster Data, http://www.asterdata.com/product/advanced-analytics.php

[7] Apache HBase, "http://hbase.apache.org/"

[8] "Data-Intensive Text Processing with MapReduce", Jimmy Lin and Chris Dyer, Morgan & Claypool Publishers, 2010.

[9] GNU Coord, http://www.coordguru.com/

[10] "The Technical Demand of Cloud Computing", Jongwook Woo, Korean Technical Report of KISTI (Korea Institute of Science and Technical Information), Feb 2011

[11] "Market Basket Analysis Example in Hadoop, http://dal-cloudcomputing.blogspot.com/2011/03/market-basket-analysis-example-in.html", Jongwook Woo, March 2011

[12] Jongwook Woo, Siddharth Basopia, Yuhang Xu, Seon Ho Kim, "Market Basket Analysis Algorithm with NoSQL DB HBase and Hadoop", The Third International Conference on Emerging Databases (EDB 2011), Korea, Aug. 25-27, 2011

[13] Jongwook Woo and Yuhang Xu, "Market Basket Analysis Algorithm with Map/Reduce of Cloud Computing", The 2011 international Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 2011), Las Vegas, July 18-21, 2011