Evaluation of Optimised Apriori Algorithm on HDFS using MapReduce in Hadoop Distributed Mode

1VVD Prasad Chelluri, 2BLVV Kumar, 3K. Purushotham Naidu, 4M. Santhosh
1,2,3 Assistant Professor
Dept. Of Computer Science Engineering,
Gayatri Vidya Parishad College of Engineering for Women,
Visakhapatnam, India

Abstract — With a revolutionary change in data analytics it requires techniques that can equally extend with the trending data processing methods. To keep in pace with this elated progress in information evaluation, calibration and storage patterns, development and implementation of large scale algorithms for data processing is gaining importance. In datamining, association rule mining and classification is a wellutilised methodology for identifying overwhelming relations from data in large scale analytics. Apriori algorithm is one such crucial algorithm to mine the frequent item sets which form the basis for finding association rules among the data. Analyzing frequent item sets is a crucial step to find rules and association between them. This stands as a primary foundation for crucial decision making. With the advent of Hadoop Map-Reduce, parallel processing and efficient memory utilisation has come into order. This paper aims to identify the potential of Apriori Algorithm which is implemented as one-phase and k-phase Apriori algorithms in MapReduce framework and further an Optimised Apriori Algorithms(OAA) has been implemented which has a full-fledged MapReduce benefits and it has been identified that Optimised Apriori Algorithm has yielded better efficiency and reduced time complexity.

Index Terms: Apriori Algorithm Optimised Apriori Algorithm, MapReduce.

INTRODUCTION

Recent technical trends in storage, processing and networking technologies lead to rapid growth of huge volumes of data in both scientific as well as commercial domains. Organizations are more inclined to make better use of this data and the data processing techniques to efficient decision making. Since the data is voluminous it requires appropriate and potential computing environments and framework to increase the precision that directly influence the decision making in real time scenario. Hadoop Framework is one such large-scale distributed batch processing infrastructure for parallel processing of voluminous data which is otherwise called as BIG DATA that flows over huge cluster of commodity computers. Hadoop is an open source project of Apache that implemented Google’s File System as Hadoop Distributed File System (HDFS) and Google’s processing framework as Hadoop MapReduce programming model. All the algorithms in this paper were implemented on Hadoop using MapReduce paradigm. MapReduce is a parallel programming model designed for parallel processing of large volumes of data by breaking the job into independent tasks across a large number of machines. MapReduce programming is inherited form the list processing languages e.g. LISP, that consists of two functions Mapper and Reducer which runs on all machines in a Hadoop cluster. The input output of these functions will be in form of <key, value> pairs. The Mapper reads the input <k1, v1>, from HDFS and produces a list of intermediate values <k2, v2>. An additional Combiner function which is optional is applied to reduce communication complexity in transferring intermediate outputs from mappers to reducers. Generally the output pairs of mapper are sorted locally and grouped on same key and applied as input to the combiner to make local sum.

With its efficient and rapid processing capabilities Hadoop has become a predominant tool for Data mining and knowledge discovery to extract useful, hidden and unknown patterns and knowledge from large database. There are many areas in datamining that generally considered for decision making. Association Rule mining is one such concept where Apriori is the basic and most popular algorithm for association rule mining proposed by R. Agrawal and R. Srikant for finding frequent itemsets based on candidate generation. Candidates are itemsets containing all frequent itemsets. The name of the algorithm Apriori is based on the Apriori property which states that all nonempty subsets of a frequent itemset must also be frequent. The core step of the algorithm is generation of candidate k-itemsets Ck from frequent (k-1)-itemsets Lk-1.[1][2][3]There has been a wide variations in the implementation of Apriori Algorithms. In this paper we have implemented an Optimised Apriori Algorithm (OAA) and evaluated its performance against the One-Phase Apriori and K-Phase Apriori algorithm where the results have evidently proven that the performance of OAA is much better when compared to the other two algorithms.[4][5][6]

Related Work

Apriori Algorithm: One-phase and K-Phase

As the outline of the paper discussed earlier, since the Apriori lacks in efficiency while dealing with the voluminous data sets and even most of the optimized techniques of Apriori using MapReduce has elated its
efficiency using multiple techniques, our work aims at evaluating the performance of the Optimised Apriori Algorithm (OAA) with two basic Apriori Models one-phase Apriori Algorithm and K-Phase Apriori Algorithm.[4][6][7][8]

In one-phase Apriori algorithm only a single phase of MapReduce task is considered for all frequent k-itemsets even though it has little implementation complexity, but the time complexity is too high making it more inefficient. Our experimental results have underlined the same effect. The following algorithm explains the implementation of the one-phase algorithm.

Map task: //one for each split
Input: S // Split i, line=transaction
Output: <key, 1>pairs, where key is an element of candidate itemsets.
1. For each transaction t in S
2. Map(line offset, t) //map function
3. For each itemset I in t /*I = all possible subsets of t */
4. Out(I,1);
5. End foreach
6. End map
7. End foreach
8. End

Reduce Task:
Input: <key2, value 2> pairs,
Minimum_support_court, where key2 is an element of the candidate k-itemset and value 2 is its occurrence in each split
Output :< key3, value3> pairs, where key3 is an element of frequent k-itemset and value3 is its occurrence in the whole dataset.
1. Reduce((key2, value2))//Reduce function
2. Sum=0;
3. While(value2 has next0)
4. Sum+=value 2 get next0;
5. End while
6. If (sum>=min_sup_count)
7. Out(key2,sum);//collected in Lk
8. End if
9. End reduce
10. End

Fig.1.One-Phase Apriori Algorithm

In the K-phase Apriori Algorithm (where k=maximum length of frequent itemsets) the algorithm needs k phases (MapReduce jobs) to find all frequent k-itemsets where phase one to find frequent 1-itemset, phase two to find frequent 2-itemset, and so on. The pseudo-code of this algorithm is shown in figure 2 and 3.

Map Task: // one for each split
Input: S; // Split i, line = transaction
Output: <key, 1> pairs, where key is an element of candidate k-itemset
1. For each transaction t in S
2. Map(line offset, t) //map function
3. For each item I in t // I = token
4. Out(I,1);
5. End foreach
6. End map
7. End foreach
8. End

Reduce Task:
Input: <key, value 2> pairs,
Minimum_support_court, where key2 is an element of the candidate k-itemset and value 2 is its occurrence in each split
Output :< key3, value3> pairs, where key3 is an element of frequent k-itemset and value3 is its occurrence in the whole dataset.
1. Reduce((key2, value2))/Reduce function
2. Sum=0;
3. While(value2 has next0)
4. Sum+=value 2 get next0;
5. End while
6. If (sum>=min_sup_count)
7. Out(key2,sum);//collected in Lk
8. End if
9. End reduce
10. End

Fig.2.Algorithm for K-phase Apriori where k=1

Proposed Algorithm: Optimised Apriori Algorithm (OAA) Taking into account the real time functioning of one-phase and k-phase Apriori Algorithm we have implemented an Optimised Apriori Algorithm which needs only two Map Reduce phases to find all frequent k-itemsets

Fig.4. Explains the data flow of our proposed OAA where each input split is assigned a mapper that employs the map function, unlike the one-phase and k-phase Apriori, in OAA the value parameter of key <Key, value> takes the entire split as input rather than the one line transaction and
further minimum support count is considered to be a value equal to the number of transactions in the input split multiplied by minimum support threshold. The map’s output is a list of intermediate key/value pairs: grouped by the key via combiner (optionally), and stored in the map worker; where the key is an element of partial frequent k-itemsets and the value is its partial count. When all map tasks are finished, the reduce task (executed by reduce worker) is started. The maps output are shuffled (fetched) to the reduce worker that calls a reduce function. The output of reduce function is a list (L_p) of key/value pairs, where the key is an element of partial frequent k-itemsets and the value equal one, stored in HDFS. Figure 5 shows the pseudo-code of this phase.

In phase two (dashed arrows in figure 6), one extra input is added to the data flow of the previous phase, which is a file (copied from Hadoop Distributed Cache, which in the stand-alone mode in local file system) that contains all partial frequent k-itemsets. The map function of this phase counts occurrence of each element of partial frequent k-itemset in the split and outputs a list of key/value pairs, where the key is an element of partial frequent k-itemset and the value is the total occurrence of this key in the split. The reduce function outputs a list (L_g) of key/value pairs, where the key is an element of global frequent k-itemsets (subset of partial frequent k-itemsets) and the value is its occurrence in the whole data set. Figure 6 shows the pseudo-code of this phase. [7],[8]

Map Task: // one for each split
Input: S_i // split i, line=transaction
Output: < key, value > pairs, where key is an element of partial frequent k-itemsets and value is its partial occurrence
1. Map(object,S_i) // Map function
2. L= apply_Apriori_on(S_i);//*Partial_min_sup_count is used*/
3. For each itemset I in L
4. Out(I, partial count);
5. End foreach
6. End map
7. End

Reduce Task:
Input: < key 2, value 2 > pairs, where key 2 is an element of the partial frequent k-itemsets and value 2 is its occurrence in each split
Output: <key 3, 1> pairs, where key 3 is an element of global candidate frequent k-itemsets
1. Reduce (key2, value2)// reduce fun.
2. Out(key2,1); // collected in L_o
3. End reduce
4. End

Fig: Pseudo Code of Phase one of Optimized Apriori Algorithm(OAA)
Reduce Task:
Input: <key2, value2>pairs, where key2 is an element of
the global candidate k-itemsets and value2 is its
occurrence in each split
Output: <key3, value3>pairs, key 3 is an element of
global frequent k-itemsets and value3 is its global
occurrence in the whole data set
12. reduce(key2,value2) // Reduce fun
13.
14. sum=0;
15. while(value2.hasNext0)
16. sum+=value2.getNext0;
17. end while
18. if (sum>=min_sup_count)
19. Out(key2,sum);// collected in LG
20. End if
21. End reduce
22. End

Fig. Pseudo code for Phase Two of Optimised Apriori Algorithm

Experimental Setup: Result Evaluation
The experimental setup has been framed by building a
Hadoop cluster that constitutes four clusters with each
cluster having 4 nodes and each node had a Ubuntu 14.04
LTS operating system with Hadoop 2.6.0 using Map
Reduce with Java 1.8.0-121.

The data set used is T1014TD100K which has been
generated by IBM’s Quest Synthetic Data Generator. The
total number of transactions are 25, 00,000 and each
transaction contains 20 items on an average. The total
number of items are 8000.The average length of frequent
Item Sets are 4.[9][10][11]

The following graphs tabulate the performance of the three
algorithms namely: one-phase,K-phase and Optimised
Apriori Algorithm (OAA)
CONCLUSION:
It has been evident from the experimental results that the implementation of Optimised Apriori Algorithm has yielded better results in all aspects when compared to One-phase and K-Phase algorithms. Even though the algorithms has put up better performance when implemented on distributed environment the optimisation has been more centric to Hadoop platform but the algorithm has evident flexibility to equip an further optimisation which can reduce the number of iterations thus making it outstandingly efficient. Further the implementation of the Optimised Apriori algorithm in Apache Spark is also anticipated to outperform the existing algorithm since Apache Spark gives more memory based computation hence reducing the complexity in iterating and I/O access.

REFERENCES:


[8] International Journal of Networked and Distributed Computing, Vol. 1, No. 2 (April 2013), 89-96 Published by Atlantis Press Parallel Implementation of Apriori Algorithm Based on MapReduce by Ning Li*.

[9] Performance Evaluation of Apriori Algorithm on a Hadoop Cluster by JA NOSILLE Department of Automation and Applied Informatics Budapest University of Technology and Economics Magyar 2: (Building Q), 1117 Budapest HUNGARY.
