Middle-East Journal of Scientific Research
24 (Recent Innovations in Engineering, Technology, Management & Applications): 69-73, 2016
ISSN 1990-9233; © IDOSI Publications, 2016
DOI: 10.5829/idosi.mejsr.2016.24.RIETMA112

An Effective Amra in Frequent Itemset from Large Transaction Dataset

G. Jayagopi and S. Pushpa

Department of CSE, St.Peter's University, Avadi, Chennai, India

Abstract: Frequent item set algorithms traverse the entire dataset in multiple passes, especially, the Apriori-like algorithms, when new candidate n-itemset is generated. It traverses the entire dataset to compute its frequency for generating frequent n-itemset and candidate (n+1)-itemset. If a dataset contains with large data size,which is inevitable for todays organization, most if not all algorithms performed not as efficient needs. We therefore attempt to solve these efficiency problems by proposing a Vertical – Apriori Map-reduce algorithm. Vertical AMRA is based on data attribute identifier which is exploited as capability metric for mining frequency item-set from large dataset in a single node that has no distributed and parallel computing system environment. Our evaluations using synthetic datasets and data from public repository suggest that Vertical AMR algorithm can offer superior efficiency in mining frequent item-sets from large transaction dataset.

Key words: Frequent item-set mining · Mapreduce · Apriori · Big data

INTRODUCTION

In Information Communication Technology and smart sensor technologies has enabled massive data being transmitted and received across commercial, industrial and health care transaction processes. Such massive amount of transactional data has embedded strategic and operational information that can be exploited by organisations management to make right time competitive advantage and sustainable growth decisions. One of the key requirements for making right time decisions is the ef?cient mining of the big data analytics. Hadoop is the computing platform that enjoys a good reputation as big data solution in recent years, since its core technique MapReduce [1] which inspired by Google MapReduce [2].

Hadoop MapReduce is a software framework for easily writing applications which process big amounts of data in-parallel on large clusters (thousands of nodes) of commodity hardware in a reliable, fault-tolerant manner.The term MapReduce actually refers to the following two different tasks that Hadoop programs perform: (i).The Map Task: This is the first task, which takes input data and converts it into a set of data, where individual elements are broken down into tuples (key/value pairs). (ii)The Reduce Task: This task takes the output from a map task as input and combines those data tuples into a smaller set of tuples. The reduce task is always performed after the map task.Typically both the input and the output are stored in a file-system. The framework takes care of scheduling tasks, monitoring them and re-executes the failed tasks. The MapReduce framework consists of a single master JobTracker and one slave TaskTracker per cluster-node. The master is responsible for resource management, tracking resource consumption/availability and scheduling the jobs component tasks on the slaves, monitoring them and re-executing the failed tasks. The slaves TaskTracker execute the tasks as directed by the master and provide task-status information to the master periodically. The JobTracker is a single point of failure for the Hadoop MapReduce service which means if JobTracker goes down, all running jobs are halted. Apriori employs an iterative approach known as level –wise search, where k0itemsets are used to explored (k+1)-itemsets. First, the set of frequent 1-itemsets is found by scanning the database to accumulate the count for each item and collecting those items that satisfy minimum support. The resulting set is denoted L1. Next, L1 is used to find L2, the set of frequent 2-itemsets, which is used to find L3 and so on, until no more frequent k-itemsets can be found. The finding of each Lk requires one full scan of the database.To improve the efficiency of the level – wise generation of frequent itemsets, an important property

Corresponding Author: G. Jayagopi, Department of CSE, St.Peter's University, Avadi, Chennai, India.

called the Apriori property is used to reduce the search space. The Apriori Principle: Any subset of a frequent itemset must be frequent. The Apriori property follows a two step process: (i) Join Step: C_k is generated by joining L_{k-1} with itself (ii) Prune Step: Any (k-1)-itemset that is not frequent cannot be a subset of a frequent k-itemset.

An interesting method in this attempt is called frequent pattern growth, or simply FP-growth, which adopts a divide-and-conquer strategy as follows. First, it compresses the database representing frequent items into a frequent pattern tree, or FP-tree, which retains the itemset association information. It then divides the compressed database into a set of conditional databases (a special kind of projected database), each associated with one frequent item or "pattern fragment," and mines each database separately. For each " pattern fragment," only its associated data sets need to be examined. Therefore, this approach may substantially reduce the size of the data sets to be searched,along with the "growth" of patterns being examined.

Vertical frequent mining is different from traditional hori-zontal frequent pattern mining method [3]. It computes the inter- sections of Transaction IDs (TID, which is identifer for each transaction) to achieve same result with Apriori-like algorithms and this method only traverses the entire dataset once, i.e one- pass. Hence lesser mining time is required. By combining vertical frequent mining method Mapreduce mechanism and Apriori we propose a new algorithm known as VAMR (Vertical Apriori Mapreduce) that is capable of mining large transactional dataset in an efficient manner within a single node [4]. VAMR algorithm will be tested using self-generated synthetic datasets as well as public dataset. We run experi- ments to support the efficient mining performance of Vertical - AMR algorithm with comparison to OPUS Miner and Apriori Mapreduce separately. Another spin-off contribution of our research is that our data generator also provides a new way to test the frequent pattern mining algorithm, since it allows us to generate different sizes of transactional datasets with the same standard format of Frequent Itemset Mining Dataset Repository.

Background

Vertical Frequent Mining: Both the Apriori and FP-growth methods mine frequent patterns from a set of transactions in TID-itemset format (i.e., {TID:itemset}), where TID is a transaction ID and itemset is the set of items bought in transaction TID. This is known as the horizontal data format. Alternatively, data can be presented in item-TID-set format (i.e., {item: TID_set}), where item is an item name and TID_set is the set of transaction identifiers containing the item. This is known as the vertical data format [5].

Mining frequent itemsets using the vertical data format. Consider the vertical data format of the transaction database, D, of Table 1. by scanning the data set once.

Table 1: Vertical Data Format of the Transaction Data Set D

Itemset	TID_set
I1	{T100,T400,T500,T700,T800,T900}
I2	{T100,T200,T300,T400,T600,T800,T900}
I3	{T300,T500,T600,T700,T800,T900}
I4	{T200,T400}
I5	{T100,T800}

Mining can be performed on this data set by intersecting the TID_sets of every pair of frequent single items [3, 6]. The minimum support count is 2. Because every single item is frequent in Table1, there are 10 intersections performed in total, which lead to eight nonempty 2-itemsets, as shown in Table 2.

Table 2: 2-Itemsets in Vertical Data Format

Itemset	TID_set
{I1,I2}	{T100,T400,T800,T900}
{I1,I3}	{T500,T700,T800,T900}
{I1,I4}	{T400}
{I1,I5}	{T100,T800}
{I2,I3}	{T300,T600,T800,T900}
{I2,I4}	{T200,T400}
{I2,I5}	{T100,T800}
{I3,I5}	{T800}

Notice that because the itemsets {I1,I4} and {I3,I5}each contain only one transaction, they do not belong to the set of frequent 2-itemset.

Based on the Apriori property, a given 3-itemset is a candidate 3-itemset only if every one of its 2-itemset subsets is frequent. The candidate generation process here will generate only two 3-itemsets:{I1,I2,I3} and {I1,I2,I5}.

Table 3: 3-Itemsets in Vertical Data Format

Itemset	TID_set
{I1,I2,I3}	{T800,T900}
{I1,I2,I5}	{T100,T800}

By intersecting the TID_sets of any two corresponding 2-itemsets of these candidate 3-itemsets,it derives Table 3. Where there are only two frequent 3-itemsets:{I1,I2,I3:2} and {I1,I2,I5:2}.

The process of mining frequent itemsets by exploring the vertical data format. First, we transform the horizontally formatted data into the vertical format by scanning the data set once. The support count of an itemset is simply the length of the TID_set of the itemset. Starting with k=1, the frequent k-itemsets can be used to construct the candidate (k+1)-itemsets based on the Apriori property.The Computation is done by interection of the TID_sets of the frequent k-itemsets to compute the TID_sets of the corresponding (k+1)-itemsets. This process repeats, with k incremented by 1 each time, until no frequent itemsets or candidate itemsets can be found.

Besides taking advantage of the Apriori property in the generation of candidate (k+1)-itemset from frequent k-itemsets, another merit of this method is that there is no need to scan the database to find the support of (k+1) - itemsets (for k=1).

This is because the TID_set of each k-itemset carries the complete information required for counting such support. However, the TID_sets can be quite long, taking substantial memory space as well as computation time for intersecting the long sets [7].

To further reduce the cost of registering long TID_sets, as well as the subsequent costs of intersections, we can use a technique called diffset, which keeps track of only the differences of the TID_sets of a (k+1)-itemset and a corresponding k-itemset. For instance, in example we have {I1}={T100,T400,T500,T700, T800,T900} and {I1,I2}= {T100, T400,T800,T900}. The diffset between the two is diffset ({I1,I2}, {I1}) = {T500,T700}. Thus, rather than recoding the four TIDs that make up the intersection of {I1}and {I2}, we can instead use diffset to record just two TIDs, indicating the difference between {I1}and {I1,I2}. Experiments show that in certain situations, such as when the data set contains many dense and long patterns, this technique can substantially reduce the total cost of vertical format mining of frequent itemsets.

Mapreduce: MapReduce allows for distributed processing of the map and reduction operations. Provided that each mapping operation is independent of the others, all maps can be performed in parallel – though in practice this is limited by the number of independent data sources and/or the number of CPUs near each source [8]. Similarly, a set of 'reducers' can perform the reduction phase, provided that all outputs of the map operation that share the same key are presented to the same reducer at the same time, or that the reduction function is associative. While this process can often appear inefficient compared to algorithms that are more sequential (because multiple rather than one instance of the reduction process must be run), MapReduce can be applied to significantly larger datasets than "commodity" servers can handle – a large server farm can use MapReduce to sort a petabyte of data in only a few hours.The parallelism also offers some possibility of recovering from partial failure of servers or storage during the operation: if one mapper or reducer fails, the work can be rescheduled – assuming the input data is still available.

Another way to look at MapReduce is as a 5-step parallel and distributed computation:

Prepare the Map() Input: the "MapReduce system" designates Map processors, assigns the input key value *K1* that each processor would work on and provides that processor with all the input data associated with that key value.

Run the User-Provided Map() Code: Map() is run exactly once for each *K1* key value, generating output organized by key values *K2*.

"Shuffle" the Map Output to the Reduce Processors: the MapReduce system designates Reduce processors, assigns the *K2* key value each processor should work on and provides that processor with all the Map-generated data associated with that key value.

Run the User-Provided Reduce() Code: Reduce() is run exactly once for each *K2* key value produced by the Map step.

Produce the Final Output: The MapReduce system collects all the Reduce output and sorts it by *K2* to produce the final outcome.

These five steps can be logically thought of as running in sequence – each step starts only after the previous step is completed – although in practice they can be interleaved as long as the final result is not affected. In many situations, the input data might already be distributed ("sharded") among many different servers, in which case step 1 could sometimes be greatly simplified by assigning Map servers that would process the locally present input data. Similarly, step 3 could sometimes be sped up by assigning Reduce processors that are as close as possible to the Map-generated data they need to process.

Proposed System: Vertical _ Apriori Mapreduce takes advantages from Apriori, Vertical frequent mining and

Mapreduce to minimize their drawbacks in order to efficiency discover frequent item-sets from large datasets [6]. Vertical-Apriori mapreduce employs the vertical frequent pattern mining to avoid the multiple scanning of entire dataset.

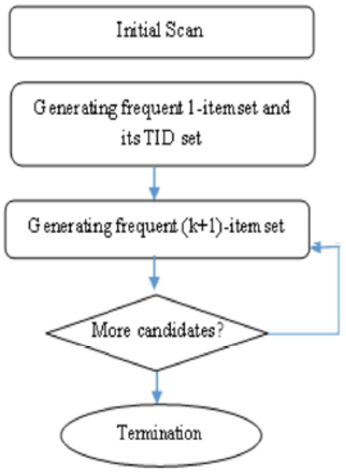
Initial Scan
Generating frequent 1-itemset and its TID set
Generating frequent (k+1)-item set
More candidates?
Termination

Fig. 1: Process of VARM

It also applies Mapreduce program paradigm on a single node in order to process large dataset in an efficient way. Apriori cooperates with vertical frequent patterns to maintain the frequent 1- itemset TID set for minimizing memory usage. The above Figure 1 illustrated the process of Vertical – Apriori Mapreduce.

```
Data: DataSet D
Map(Key,Value-list)
foreach Record ri in D do
    foreach Item I in ri do
        output(I,1)
    end
end
```

Algorithm 1: Initial Map

```
Data: Key-pairs KP(Key,Value-list)
Reduce(Key,Value)
foreach Key k in KP do
    foreach value in ks value list do
        k.value = k’s size of value list
        if k.value = minimum supportthen
            output(k,k.value)
        end
    end
end
```

Algorithm 2: Initial Reduce

Algorithmj 1 and 2 showed the fundamental of the Mapreduce algorithm for initial scan.

```
Data: Candidate Itemsets C
Map(Key,Value-list)
foreach Candidate c in C do
    foreach Item i in c do
        c.value-list += intersection(i.TIDSet)
    end
end
```

Algorithm 3: Main Map Process

```
Data: Key-pairs KP(Key,Value-list)
Reduce(Key,Value)
foreach Key k in KP do
    foreach value in ks value list do
        k.value = k’s size of value list
    if k.value-list.size = minimum
        supportthen output(k,k.value)
 end
    end
end
```

Algorithm 4: Main Reduce Process

```
Data: Key-pairs KP(Key,Value-list)
NextCandidate(new keys)
foreach Key k in KP do
    foreach k(i + 1) in KP do
        new keys +=join(ki,k(i + 1))
    end
end
```

Algorithm 5: NextCandidate Process

```
Data: Key-pairs KP(Key,Value-list)
Main()
Inital Scan
Candidate = NextCandidate(Reduce(Map(Dataset D)))
while !candidate.isEmpty do
    Candidate = NextCandidate(Reduce(Map(Dataset
    D)))
end
```

Output File: Algorithm 3,4,5 and 6 demonstrated the entire Vertical-Apriori Mapreduce algorithm. Use above Table1as an example,our minimum support is 50%, so after initial scan, the frequent 1-itemset and its TID would be Then, the nextCandidate will take these keys as input to generate next candidate.

Experiments: In our experiments, this Vertical-Apriori Mapreduce algorithm is implemented in Java and all experiments are running on the Windows platform with core i5-2520M and 4GB RAM.Datasets used for experiments including datasets from public repository (frequent itemset dataset repository) and self-generated datasets with different number of records and different number of attributes. In Experiment, we compared the Vertical-AprioriMapreduce with Apriori Map reduce in a distributed system using dataset from Frequent Itemset dataset repository T10I1D100K. The results are as shown in the Figures 4 and 5. Comparing with results, our results are much more efficient. With the increased minimum support, the performance on this dataset is rather stable, ranging from 0.53 seconds to 0.56 seconds. Apriori Map reduce takes longer time to process this dataset, its run time decreased as when minimum support increased [9].

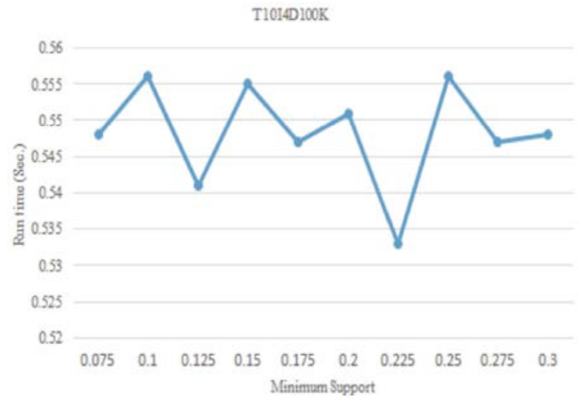

Fig. Result of VAMR T10I4100K

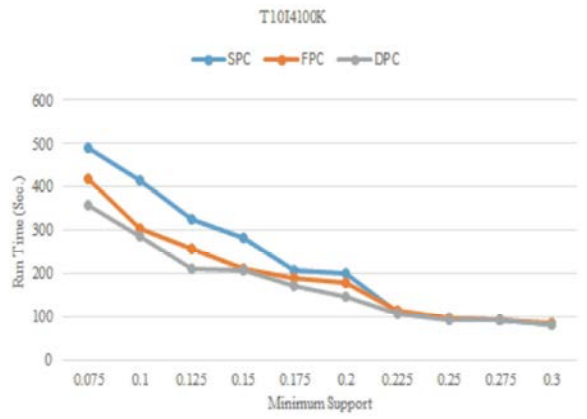

Fig. Result of Apriori Mapreduce of T1014100K

CONCLUSION

In this paper an Apriori effective Mapreduce algorithm and evaluated its performance using public repository dataset and self –generated datasets. Given evaluation results demonstrated AMR algorithm is capable to process large dataset in an efficient manner, compared with OPUS Miner and Apriori Mapreduce. Vertical-Apriori Mapreduce is more efficient. This proposed algorithm adopted the advantages from Apriori, Vertical frequent pattern mining and Mapreduce to minimize the demerits enables a single node to process large dataset in an efficient manner.

REFERENCES

1. H. Inc., Hadoop, Inc., Hadoop.
2. Dean, J. and S. Ghemawat, 2010. "Mapreduce: a flexible data processing tool", Communcat ions of the ACM, 53(1): 72-77.
3. Mueller, A., 1998. "Fast sequential and parallel algorithms for association rule mining: A Comparision,".
4. Frequent Itemset Dataset Repository, 2013.
5. Pasquier, N., Y. Bastide, R. Taouil and L. Lakhal, 1999. "Efficient mining of association rules using closed itemset lattices," Information Systems, 24(1): 25-46.
6. Zaki, M.J. and C.J. Hsiao, 2002. "Charm: An Efficient algorithm for closed itemset mining," in SDM, SIAM, 2: 457-473.
7. Han, J., J. Pei and Y. Yin, "Mining frequent patterns without candidate generation," in ACM SIGMOD Record, 29(2). ACM.
8. Zaki, M.J., 1999. "Parallel and distributed Association mining: A survey," IEEE Concurrency, 7(4): 14-25.
9. Agrawal, R., H. Mannila, R. Srikant, H. Toivonen, A.I. Verkamo *et al*., 1999. "Fast discovery of association rules," Advances in Knowledge Discovery and Data Mining, 12(1): 307-328.