Comparison of Apriori and Parallel FP Growth over Single-node and Multi-node Hadoop Cluster

Chandanmeet Narula, Geeta Sikka

***Abstract*—Frequent itemsets play a fundamental role in finding fascinating patterns in databases, thus helping in many data mining tasks. It helps to identify set of items, characteristics, symptoms etc. that very commonly occur together in our database. To find these itemsets, the typical algorithms known to us are Apriori algorithm and FP (Frequent Pattern) Growth algorithm. In this paper, we implemented these two algorithms over Hadoop MapReduce platform and compared the execution time of both the algorithms. We found that over Hadoop platform also, FP Growth performs better than Apriori algorithm.**

Index Terms: Frequent Itemsets, Apriori, FP Growth, Hadoop MapReduce, Comparison

I. INTRODUCTION

Today we are in the age of information. We are collecting tremendous amount of information from various sources with the help of our sophisticated technologies such as computers, satellites etc. The information is in the form of business transactions, scientific/personal/medical data, digital media and many more forms. In this information age, this information leads to power and success. But this enormous information simply does not lead to decision making. There is need to develop powerful means for analyzing and interpreting the data in order to extract interesting knowledge from this which can be helpful in taking decisions. So, here Data Mining and its techniques come into picture.

The organization of paper is stated as: Section 2 puts light on basic concepts about Association Mining which is one of the data mining techniques. Under it, we will see the two popular mining algorithms: Apriori and FP Growth algorithm. Section 3 will give brief idea about Hadoop and Map-Reduce Approach. Section 4 presents the literature survey done. In section 5, we will see Apriori and Parallel FP Growth algorithms over Map-Reduce and we will make the comparison of the execution time of two algorithms over Hadoop Map-Reduce Platform. Finally, the section 6 derives the conclusion of paper.

Chandanmeet Narula is pursuing M Tech from NIT Jalandhar, India and the email id is: chandannarula20@gmail.com

Geeta Sikka is Associate Professor and Head of Computer Science Department at NIT Jalandhar, India and email id is: sikkag@nitj.ac.in

II. BASIC CONCEPTS

First, Association Rule Mining is used to discover relations between various items in large databases [10] using different measures such as minimum support, confidence and other measures. Association rules were introduced for discovering patterns between items in large-scale transaction data by Rakesh Agrawal et al. [10]. These association rules are used in many applications such as market-basket analysis, web usage mining, in unmasking the intrusions, stable production and bio-informatics etc.

Association Rule Mining is a two-step process:

- Generation of Frequent Item-sets: Generate all item-sets whose support ≥ minimum support minsup.
- Generation of Rules: The association rules are generated using the generated frequent item-sets. They should satisfy the minimum support and minimum confidence criteria.

Two Important Algorithms for Association Rule Mining are:

A. Apriori Algorithm

It follows Apriori principle which states that the subsets of frequent itemsets are also frequent. Say itemset: {PQR} is frequent, it means: P, Q, R, PQ, QR, and PR are also frequently occurring in transactional database

- In the first pass of the algorithm, the frequency of occurrences of each item is counted and frequent 1-itemset is determined. [10].
- The next pass of the algorithm, say pass k, consists of two phases:
  - The frequent itemsets of previous pass are joined with itself to find candidate itemsets for the next pass using the Apriori Candidate Generation Function [10].
  - Then, we need to check which of the itemsets out of all candidate itemsets are frequent, hence the database is scanned and the support of candidate itemsets is found out.

The flow chart for Apriori algorithm is shown in Fig 1.

B. Frequent Pattern (FP) Growth Algorithm

Here, frequent itemset mining is possible without the generation of candidates. Only two scans of database are needed. It consists of two steps:

- FP tree is built which is compact in size and for its construction; the database is scanned only twice.
- Once the FP tree has been constructed, the frequent patterns will be extracted from it.

ISBN: 978-988-14047-1-8
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

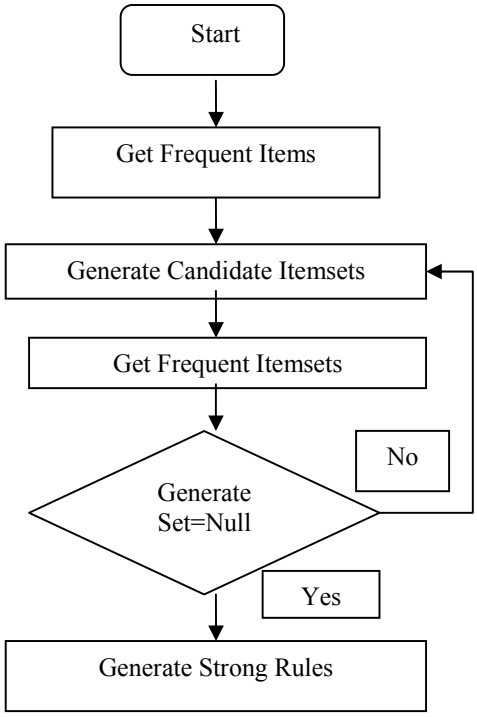

Fig. 1. Flow chart of Apriori Algorithm

This has been outlined below with the help of example where Table I shows the list of transactions in our transactional database. Table II contains the support count of all the items present in the input database. Table III shows the transactions sorted according to the frequency of items. The threshold i.e. minimum support count is 2. So the items/itemsets whose occurrence count is less than 2 are removed from the frequent item list. Fig. 2 shows the constructed FP tree. From this generated FP-tree, conditional pattern base, conditional FP-trees are created from which the frequent patterns are generated as shown in Table IV.

These algorithms are applied over large data sets. These huge datasets present new challenges such as data storage and data transfer. To manage the data resources and data flow between the storage and computing resources is becoming the mail bottleneck, that too on a single sequential machine. The solution for the above problem is parallel and distributed computing. Here, Hadoop comes into picture which provides parallel computation and is used to deal with Big Data.

C. Big Data And Hadoop

Each day, the huge amount of data is getting generated by various sources such as climatic sensors, social media, purchase transaction records etc. and this data is accounting to 2.5 quintillion bytes per day. The facts state that 90% of this data has been produced in just last two years. Due to volume, variety of this data, it is termed as big data [11]. This is not just the matter of size of data that is being generated; rather it helps to gain deep understanding of new and originating data that can help in making the businesses more agile, and finding answers to questions that seemed impossible in the past.

Table I: Transactional Database

TIDs	List of Items
1	a b e
2	b d
3	b c
4	a b d
5	a c
6	b c
7	a c
8	a b c e
9	a b c

Table II: Frequent of Items

b	7
a	6
c	6
d	2
e	2

Table III: Transactions order according to Frequency of Items

TIDs	List of Items
1	b a e
2	b d
3	b c
4	b a d
5	a c
6	b c
7	a c
8	b a c e
9	b a c

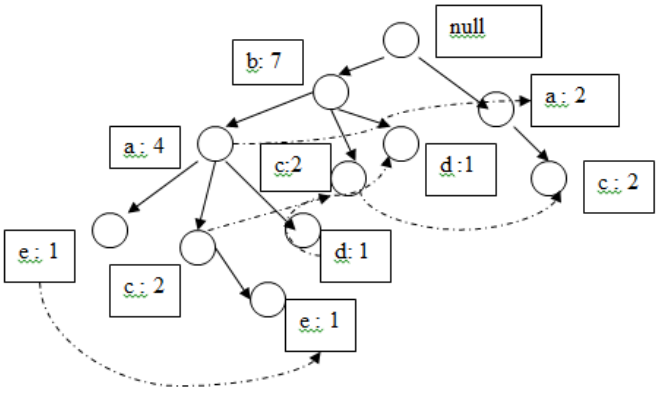

Fig. 2. FP Tree Construction

ISBN: 978-988-14047-1-8
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

Table IV: Conditional Pattern Base, Condition FP tree and Frequent Pattern Generation from the constructed FP-tree

Item	Conditional Pattern Base	Condition FP-tree	Frequent Patterns Generated
e	{(ba:1), (bac:1)}	{b:2, a:2}	be:2, ae:2, bae:2
d	{(ba:1),(b:1)}	{b:2}	bd:2
c	{(ba:2),(b:2),(a:2)}	{b:4, a:2} ,{a:2}	bc:4, ac:2, bac:2
a	{(b:4)}	{b:4}	ba:4

Hadoop is the heart of platforms for assembling Big Data now-a-days. It uses a distributed computing architecture that consists of multiple servers installed on commodity hardware, thus making it very inexpensive to scale and support extremely large data stores. [12] Hadoop is moreover an open source framework. We can say that Hadoop provides a reliable, scalable platform for storage and analysis. Map-Reduce is a batch query processor. It runs an adhoc query on your whole dataset and generates the result in reasonable time. It provides programming model that abstracts problem from disk read and writes and transforms into a computation over a set of keys and values. You just need to write your program in terms of Map and Reduce functions and the input data should have the form of key-value pair. The map function processes a key/value (K1,V1) pair to generate a set of intermediate key/value pairs (K2,V2), and the reduce function merges all intermediate values associated with the same intermediate key and outputs a new set of key/value pair(K3,V3) [13]. The processing will be taken care of by the Hadoop Platform.

In Hadoop, the MapReduce system reads the input and writes the final results from/into its file system (HDFS). There is a job tracker that runs on the master node of Hadoop cluster taking care of the progress of the job and there are task trackers that run on worker nodes that perform map and reduce tasks in real.

map(K1, V 1) → [<K2, V2>]

reduce(K2, {V 2}) → [<K3, V3>]

The working of Hadoop has been depicted in Fig. 3.

III. RELATED WORK

We studied the previous work done regarding our concern i.e. the implementation of Apriori and Parallel FP Growth algorithm in general and over map reduce and below are our findings:

Rahul Mishra et al. [1] have applied Apriori algorithm and FP growth over web mining data in order to determine the web usage of any site to determine the factors why users stay on a particular site, what is the pattern of the site usage by them. This is helpful in increasing the sales on E-Commerce site.

Othman Yahya et al. [2] suggest using Hadoop Map Reduce Programming model for parallel and distributed computing. It is an effective model to write easy and efficient applications where large datasets can be processed on clusters of computing nodes, that too in a fault tolerant manner. This paper provides insights into the implementation of Apriori over Map Reduce model and the researchers proposed new algorithm MRApriori: MapReduceApriori Algorithm.

Juan Li1 et al. [3] have implemented Apriori algorithm over Amazon EC2 Map Reduce Platform. The paper clearly explains how things work over EC2 cloud. The researchers implemented revised Apriori i.e. changing Apriori algorithm as per MapReduce platform over single node and multiple node Hadoop cloud.

Ning Li et al. [4] proposed parallel Apriori algorithm over MapReduce. In this algorithm, the occurrence of each potential candidate of size k is counted by the map function and thus there is parallel computation of all the potential candidates during this map stage. Then, the reduce function performs the procedure of summing the occurrences counts. For each round of the iteration, such a job is carried out to implement the occurrences computing for potential candidates of size k.

Zahra Farzanyar et al. [5] told that being Apriori a serial mining algorithm, it has to be converted into parallel mining algorithm. Many parallel apriori algorithms were introduced but new problems that didn't exist in sequential computing came into picture with these parallel apriori algorithms such as: balancing the work load, partitioning of data and distribution of the jobs and their assignment to nodes and parameters passing between nodes. To encounter these problems, MapReduce model was introduced. MRApriori Algorithm based on MapReduce model outperforms other parallel apriori algorithms, but still the count of partial frequent itemsets generated is large. The authors of this paper advised Improved MPApriori algorithm and the results proved that the execution time was considerably reduced.

Zahra Farzanyar et al. [6] extended their work which they presented in their previous paper [5].They made changes in phase I of Apriori Algorithm and produced data containing partial frequent item sets count and this is given as input to phase II. So, the numbers of itemsets to be taken care of by Map function is considerably reduced in phase II and hence work of Reducers is also decreased as compared to the phase II of IMRApriori algorithm. Thus, communication load between mappers and reducers in phase II is decreased and execution times is also reduced.

In [7], Haoyuan Li et al. thought to parallelize FP growth to achieve greater speed up since the sequential FP growth is much faster than Apriori algorithm. They used this approach for query recommendation over web and tried to understand how Google fetch results.

Le Zhou et al. [8] advised balanced parallel FP Growth algorithm over MapReduce approach in their work. It considers the load balance feature which helps in improving parallelization and it leads to increase in performance.

ISBN: 978-988-14047-1-8
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

Sankalp Mitra et al. [9] proposed improved parallel FP Growth algorithm where they considered the fact that the efficiency of MapReduce model significantly reduces when the operations have to be performed over small files. They generate large sequence file by merging small files to form a large transactional database and then applied PFP algorithm over this one large file.

IV. ALGORITHMS AND THEIR COMPARISON

We were curious to know how these two popular mining algorithms work over MapReduce. We had implemented these algorithms in Java on a sequential machine earlier, but not on Hadoop MapReduce Platform. It was interesting to learn how the sequential algorithms are converted into parallel algorithms. The Mapper and Reducer functions had to be defined for them.

For implementing Apriori Algorithm over Hadoop MapReduce, the below given algorithms have been followed by us:

A. Apriori Algorithm over MapReduce

- To generate 1-FrequentItemset, follow algorithms have been followed by us.

Procedure: **Mapper** (key, value=$Trans_i$)
foreach item $aItem_i$ in $Trans_i$ **do**
  Call Output ($aItem_i$, 1);
end
Procedure: **Reducer**(key=$aItem_i$, value=$S(aItem_i)$)
Count← 0;
foreach item 1 in $Trans_i$ **do**
  Count← Count + 1;
end
if Count > minSupport **then**
  Call Output ($S(aItem_i)$, Count);
end

- After 1-FrequentItemset has been generated, for each level k, following mapper and reducer function is used to generate k-FrequentItemset

Procedure: **Mapper**(key, value=$Trans_i$)
Generate the k^{th} itemset from output of last level k-1
Generate Candidate Itemset,C_k from k-1 itemset
foreach candidateItem $cItem_i$ in C_k **do**
  if $Trans_i$ contains $cItem_i$ **then**
    Output ($cItem_i$, 1);
  end
end
Procedure: **Reducer**(key=$aitem_i$, value=$S(aItem_i)$)
Count ← 0;
foreach item 1 in $Trans_i$ **do**
  Count ← Count + 1;
end
if Count > minSupport **then**
  Call Output ($S(aItem_i)$, Count);
end

B. Parallel FP Growth Algorithm over MapReduce

We have implemented Parallel FP Growth algorithm following the paper written by Haoyuan Li et al. [7] where the whole process has been divided into 5 steps:

- Step 1. Sharding of database
- Step 2. Parallel counting of items
- Step3. Grouping items
- Step4. Parallel FP Growth
- Step 5. Aggregating

The brief outline of the above steps is:

1. Parallel Counting: MapReduce pass is performed to calculate the support value of all items in the database. Each mapper is given one slice or we can say one shard of database and it gives count of 1 to each item appearing in the transaction. The output of Mapper is given to Reducer where the results are combined of all Mappers and at the end of this MapReduce pass, we have item list of the transactions, i.e. we come to know what all items are in our database and the count of occurrence of these items. The output of reducer is stored in F [] list.
2. Parallel FP Growth: It's the key step of PFP algorithm. This step takes one MapReduce pass where mapper and reducer perform following functions: Mapper deals with generation of group dependent transactions and Reducer performs FP Growth on group dependent shards given as output by Mapper. In this phase, the mapper produces key-value pairs and in the reducer process, the local FP trees and conditional FP trees are generated recursively, considering the minimum support count.
3. Aggregating: The output of Parallel FP Growth reducer phase is aggregated to get the final results

Aggregating Algorithm

We made changes in the Aggregating algorithm (rest of the algorithms have been implemented as described in [7].) and implemented it as per below given algorithms.

Procedure: **Mapper**(key, value=v + supp(v))
foreach first item $aItem_i$ in v **do**
  Call Output($aItem_i$, v + supp(v));
end
Procedure: **Reducer**(key=$aItem_i$, value=S(v + supp(v)))
Define HashMap to store unique patterns : MAP<pattern,support>;
foreach pattern v in v + supp(v) **do**
  if MAP contains pattern v **then**
    if supp(MAP(v)) < supp(v) **then**
      insert and replace <v,supp(v)> in MAP;
    end
  else
  insert <v,supp(v)> in MAP;
  end
end
foreach entry <v,supp(v)> in MAP **do**

ISBN: 978-988-14047-1-8
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

Call Output(v, supp(v));

end

We performed the experiment over single node Hadoop cluster installed over Ubuntu operating system having specifications Intel(R) Core(TM)2Duo CPU @ 2.2 GHz and 3 GB RAM and with the transactions of database T10I4D100K and produced the results shown in (Table V).

We also performed this experiment over 4-node Hadoop cluster installed over Ubuntu operating system having specifications Intel(R) Core(TM) i5-3330 CPU @ 3.00GHz and 8 GB RAM and found the results given in Table VI & Fig 4.

V. CONCLUSION

So, in our work, we find that as the number of transactions is increasing, the time being taken to generate frequent itemsets is also increasing. The motive of paper was to compare the time taken in generation of frequent itemsets by two popular algorithms: Apriori and FP Growth over Map-Reduce model and we conclude that FP Growth algorithm outperforms Apriori over Hadoop Map-Reduce Platform. And we also find that over multi-node Hadoop cluster, time is noticeably reduced for finding the frequent itemsets from given set of transactional database. We also notice that the processor and the memory available also play a major role in getting speedy results.

REFERENCES

[1] Rahul Mishra, Abha choubey, "Comparative Analysis of Apriori Algorithm and Frequent Pattern Algorithm for frequent Pattern Mining in Web Log Data" , *In International Journal of Computer Science and Information Technologies*, Vol. 3 (4) , 2012.

[2] Othman Yahya, Osman Hegazy, Ehab Ezat, "An Efficient Implementation of Apriori Algorithm Based on Hadoop-MapReduce Model", In *International Journal of Reviews in Computing,* Vol. 12 2012.

[3] Juan Li , Pallavi Roy , Samee U. Khan , Lizhe Wang , Yan Bai, "Data Mining Using Clouds: An Experimental Implementation of Apriori over MapReduce", In *Proceedings of 12th International Conference on Scalable Computing and Communications (ScalCom'13)*, 2012

[4] Ning Li, Li Zeng, Qing He and Zhongzhi Shi, "Parallel Implementation of Apriori Algorithm Based on MapReduce", In *Proceedings of 13th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing*, 2012.

[5] Zahra Farzanyar, Nick Cercone, "Efficient Mining of Frequent itemsets in Social Network Data based on MapReduce Framework", In *IEEE/ACM International Conference on Advances*, 2013.

[6] Zahra Farzanyar, Nick Cercone, "Accelerating Frequent Itemsets Mining on the Cloud: A MapReduce –BasedApproach", In *Proceedings of* 13th International Conference on Data Mining Workshop, 2013.

[7] Haoyuan Li, Yi Wang, Dong Zhang, Ming Zhang, Edward Chang, "PFP: Parallel FP-Growth for Query Recommendation", In *Proceedings of the 2008 ACM conference on Recommender systems,* 2008.

[8] Le Zhou, Zhiyong Zhong, Jin Chang, "Balanced Parallel FP-Growth with MapReduce", In *Information Computing and Telecommunications (YC-ICT),* 243-246. 2010

[9] Sankalp Mitra, Suchit Bande, Shreyas Kudale, Advait Kulkarni, Leena A. Deshpande, "Efficient FP Growth using Hadoop-(Improved Parallel FP-Growth)", In *International Journal of Scientific and Research Publications*, Vol. 4, Issue 7, 2014.

[10] Rakesh Agrawal and Ramakrishnan Srikant, "Fast Algorithms for Mining Association Rules", http://rakesh.agrawal-family.com/papers/vldb94apriori.pdf, 1994

[11] Bringing big data to the enterprise. #ibmbigdata DOI= https://www-01.ibm.com/software/in/data/bigdata

[12] What is Big Data? DOI= http://www.zettaset.com/index.php/info-center/what-is-big-data/

[13] Hadoop: MapReduce Tutorial, DOI= https://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html

ISBN: 978-988-14047-1-8
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

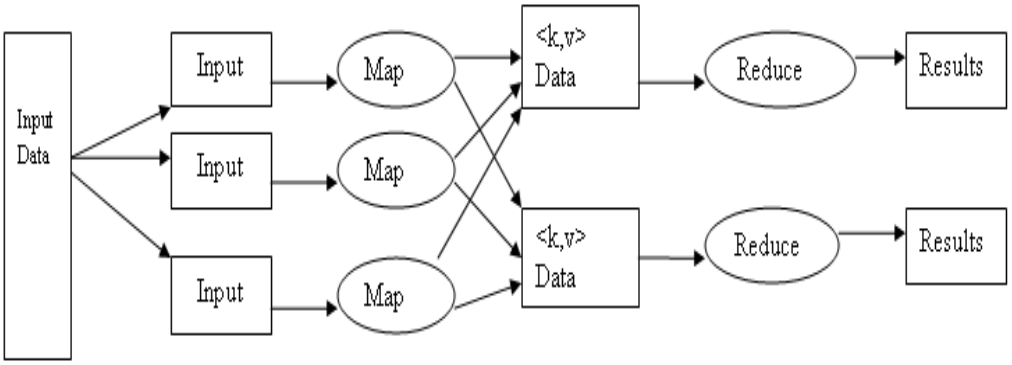

Fig. 3. Processing of data by Hadoop MapReduce

Table V: Comparison of Apriori and Parallel FP Growth Algorithm over single-node Hadoop Cluster

Number of Transactions	Apriori Algorithm (Time in secs)	PFP Growth Algorithm (Time in secs)
2000	78.340	33.599
5000	216.492	81.694
9000	603.050	384.879
10000	661.511	626.736
15000	1311.729	1035.475
20000	3971.819	3392.179
25000	6368.553	6227.533

Table VI: Comparison of Apriori and Parallel FP Growth algorithm over 4-node Hadoop Cluster

Number of Transactions	Apriori Algorithm (Time in secs)	PFP Growth Algorithm (Time in secs)
1500	92.054	45.84
5000	154.607	54.87
10000	250.673	101.943
15000	377.843	164.004
20000	600.154	261.13
25000	916.679	366.267

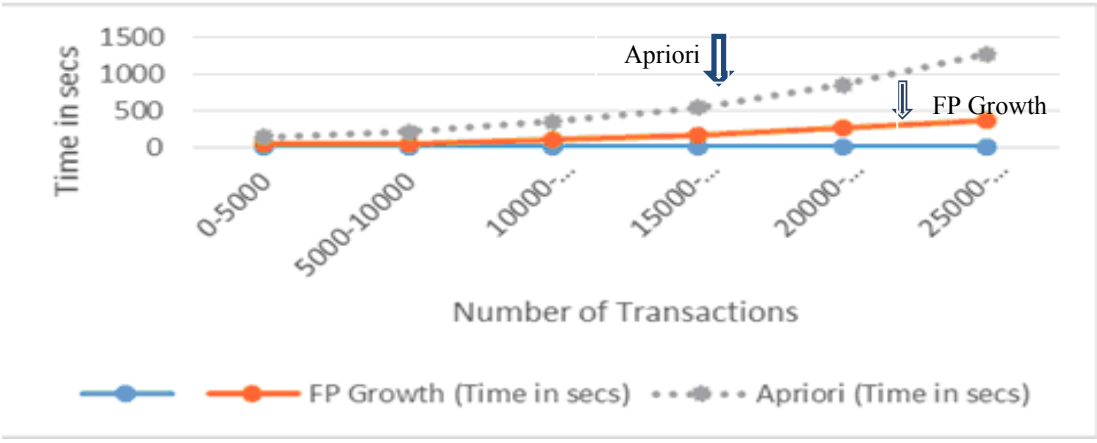

Fig. 4. Graphical Representation of Apriori and PFP Growth Performance over 4-node Hadoop Cluster

ISBN: 978-988-14047-1-8
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)