ISSN 2395-1621

# Improving Efficiency of Parallel Mining of Frequent Itemsets using Fidoop-hd

#1Anjali Kadam, #2Nilam Patil

1mianjalikadam@gmail.com
2snilampatil2012@gmail.com

#12Department of Computer Engineering

Pune University, Maharashtra, India.

## ABSTRACT

**Present parallel mining algorithms for frequent item sets lack a mechanism that enables automatic parallelization, load balancing, data administration, and fault liberality on large clusters. As a solution to this problem, design a parallel frequent item sets mining algorithm called Fidoop using the Map Reduce programming model. To achieve compressed storage and avoid building restrictive pattern bases, Fidoop incorporates the frequent items ultrametric tree, rather than regular FP trees. In Fidoop, two MapReduce jobs are implemented to complete the mining task. In the complex MapReduce job, the mappers independently decay item sets, the reducers perform combination operations by compressing data. This system implement Fidoop on private Hadoop cluster. This system show that Fidoop on the cluster is sensitive to data distribution and dimensions, because item sets with distinct lengths have different decaying and construction costs. In this paper system improve Fidoop's performance, system develop a workload balance metric to measure load balance across the cluster's computing nodes. System processes Fidoop-HD, an extension of Fidoop, to speed up the mining performance for high-dimensional data analysis. Experiments are shown using real-world celestial face of data demonstrate that our proposed solution is efficient and scalable.**

**Keywords: Frequent item sets, Hadoop cluster, load balance, MapReduce.**

ARTICLE INFO

**Article History**

Received: 9th July 2016

Received in revised form :

9th July 2016

Accepted: 12th July 2016

**Published online :**

**12th July 2016**

## I. INTRODUCTION

Data Mining has a great approach about the information industry and in society as a whole in recent years. One of the major problems in data mining is finding association rules from databases of transactions where each transaction consists of a set of items. Many ways have been proposed to find frequent item sets from a large database. Parallel computing offers a possible solution to the computation requirement of this task, if the efficient and scalable parallel algorithms can be designed.

The rest of paper is organized as follows sections. Section II describes research done available in this domain. In Section III, systems have discussed the System Overview and other. Section IV describes conclusion and future work

## II. RELATED WORK

Apriori is a classic algorithm using the generate-and-test process that generates a large number of candidate itemsets ; Apriori has to repeatedly scan entire database[1] .To reduce the time required for scanning databases, Han et al. proposed a novel approach called FP-growth, which avoids generating candidate itemsets[2]. Most previously developed parallel FIM algorithms were built upon the Apriori algorithm. Unfortunately, in Apriori-like parallel FIM algorithms, each processor has to scan a database multiple times and to exchange an excessive number of candidate itemsets with other processors[3].

Therefore, Apriori-like parallel FIM solutions suffer potential problems of high I/O and synchronization overhead, which make it strenuous to scale up these parallel algorithms. This system will implement the parallel Apriori algorithm based on MapReduce, which makes it applicable to mine association rules from large databases of transactions [4].

## III. IMPLEMENTATION DETAILS

In this a parallel frequent itemsets mining algorithm called FiDoop (Frequent Itemsets in Hadoop) using the

MapReduce programming model is proposed to achieve compact storage and avoid building conditional pattern bases.

## IV. Proposed System Overview

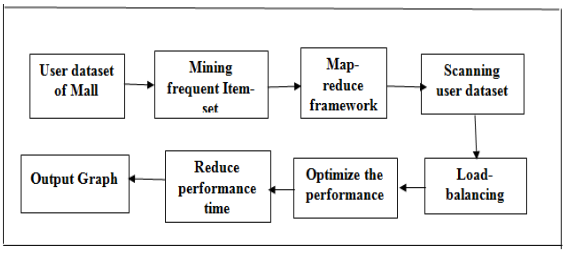
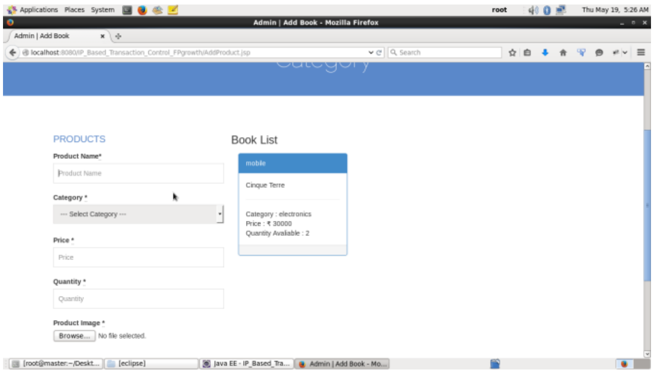


Fig.1. System overview diagram

### User Mall dataset:

In the first step, system will take as a input mall dataset. In this system are using e-shopping mall dataset. Whereas fields are like userid, productid, transactionid, categoryid etc. The dataset is prepared here for more clear idea of high dimensional data by user. This data generated by user entry. This system getting more data by varied user. This system provides opportunity to user. The Dataset is


Applications Places System
root
Thu May 19, 5:26 AM
Admin | Add Book - Mozilla Firefox
Admin | Add Book
localhost:8080/IP_Based_Transaction_Control_FPgrowth/AddProduct.jsp
Search
[root@master:~/Deskt...
[eclipse]
Java EE - IP_Based_Tra...
Admin | Add Book - Mo...

PRODUCTS

Product Name* Product Name

Category * --- Select Category ---

Price * Price

Quantity * Quantity

Product Image * Browse... No file selected.

Book List

mobile

Cinque Terre

Category : electronics
Price : ₹ 30000
Quantity Avaliable : 2

Fig2. Data set Entry Form.

referred for this system is form this link *https://**mahout**.apache.org/users/basics/collections.html*

### Mining frequent itemsets:

In this phase system discover way to discover right information from huge data. Extracting frequent patterns from a large transaction database. A typical ARM application is market basket analysis. An association rule, for example, can be “if a customer buys *A* and *B*, then 90% of them also buy *C*.” In this example, 90% is the confidence of the rule.

The ARM process can be decomposed into two phases: 1) identifying all frequent itemsets whose support is greater than the minimum support and 2) forming conditional implication rules among the frequent itemsets. The first phase is more challenging and complicated than the second one[5].

In this paper system take input as text file. This file contains structured and unstructured data fields. This text file contains data about e-shopping mall.

In this system by using rule mining generates relations among items which generates fix candidate itemsets. Where these candidate itemsets are used to for foremost mining so it reduce time and increase efficiency of algorithm.

### MapReduce Framework:

MapReduce is a promising parallel and scalable programming model for data-intensive applications and scientific analysis. A MapReduce program expresses a large distributed computation as a sequence of parallel operations on datasets of key/value pairs. A MapReduce computation has two phases, namely, the Map and Reduce phases. The Map phase splits the input data into a large number of fragments, which are evenly distributed to Map tasks across the nodes of a cluster to process. Each Map task takes in a key-value pair and then generates a set of in-between key-value pairs. After the MapReduce runtime system groups and sorts all the middle values associated with the same middle key, the runtime system delivers the in-between values to Reduce tasks. Each Reduce task takes in all in-between pairs associated with a particular key and emits a final set of key, value pairs. Both input pairs of Map and the output pairs of Reduce are managed by an underlying distributed file system[10].

MapReduce has been widely adopted by companies like Google, Yahoo, Microsoft, and Facebook.

Hadoop—one of the most popular MapReduce implementations—is running on clusters where Hadoop distributed file system (HDFS) stores data to provide high aggregate I/O bandwidth. At the heart of HDFS is a single NameNode—a master server that manages the file system namespace and regulates access to files. The Hadoop runtime system establishes two processes called JobTracker and TaskTracker. JobTracker is responsible for assigning and scheduling tasks; each TaskTracker handles Map or Reduce tasks assigned by JobTracker.

In light of the MapReduce programming model, system design a parallel frequent itemsets mining algorithm this algorithm called Fidoop. The design goal of Fidoop is to build a mechanism that enables automatic parallelization, load balancing, and data distribution for parallel mining of frequent itemsets on large clusters [9].

Apache Pig, is platform analyzing large datasets. Pig’s language, Pig Latin, is simple query algebra. Pig is simple data flow language used in analyzing large datasets. Pig scripts are automatically converted into MapReduce jobs by the Pig interpreter, so you can analyze the data in hadoop cluster.

Pig provides feature of User defined function as a way to specify custom processing.

### Scanning User dataset

In this phase, system scans user dataset as follows,

1) The first phase involving two rounds of scanning a database is implemented in the form of two MapReduce jobs. The first MapReduce job is responsible for the first round of scanning to create frequent oneitemsets. The second MapReduce job scans the database again to generate *k*-itemsets by removing infrequent items in each transaction.

2) The second phase involving the construction of a *k*-itemsets and the discovery of frequent *k*-itemsets is handled by a second MapReduce job, in which *h*-itemsets ($2 \leq h \leq M$) are directly decomposed into a list of ($h - 1$)-itemsets, ($h - 2$)-itemsets, . . . , and twoitemsets.

In the second MapReduce job, the generation of short itemsets is independent to that of long itemsets.

In other words, long and short itemsets are created in parallel by our parallel algorithm.

The two MapReduce jobs of our proposed Fidoop are described in detail. The first MapReduce job discovers all frequent items or frequent one-itemsets. In this phase, the input of Map tasks is a database, and the output of Reduce tasks is all frequent one-itemsets. The second MapReduce job scans the database to generate *k*-itemsets by removing infrequent items in each transaction. First, Fidoop handles a set of mappers, each of which repeatedly decomposes *h*-itemsets $(2 < h \leq M)$ into a list of $(h - 1)$-itemsets, $(h - 2)$-itemsets, . . . , and two-itemsets. Applying multiple mappers to decompose *h*-itemsets in parallel improves data storage efficiency and I/O performance[10].

Then, Fidoop leverages multiple reducers to merge itemsets with the same item number including original itemsets. Fidoop's tree-construction procedure takes full advantage of the Hadoop runtime system, in which the shuffling phase copies result pairs with a same key generated by mappers to a reducer. In this approach second MapReduce job, system chooses item numbers as output keys of key-value pairs emitted by the mappers. The following preliminary findings motivate us to address a pressing issue pertinent to balancing load in Fidoop: 1) large itemsets give rise to high-decomposition overhead and 2) and small decomposed itemsets lead to a large number of itemsets. To achieve good load balancing performance, system incorporate constraints in the shuffling phase of the MapReduce jobs in Fidoop, thereby balancing the number of itemsets across reducers[13]

## Load Balance:

In this phase decomposition process starts, If the length of an itemset is *m*, the time complexity of decomposing the itemset is *O(*2*m)*. Thus, the decomposition cost is exponentially proportional to the itemset's length. In other words, when the itemset length is going up, the decomposition overhead will dramatically enlarged. The data skewness problem is mainly induced by the decomposition operation, which in turn has a significant performance impact on Fidoop[2]. The first step toward balancing load among data nodes of a Hadoop cluster is to quantitatively measure the total computing load of processing local itemsets. System achieve this first step by developing a workload-balance metric to quantify load balance among the data nodes. System consider a database partitioned across *p* data nodes. Let $p_i(\mathrm{IS_m}) = (C_i(\mathrm{IS_m})/\sum_{j=1}^{p} C_j(\mathrm{IS_m}))$ be the probability that node *i* contains itemsets whose length is *m*. Here, IS*m* denotes a set of itemsets where the length of each itemset is *m*; $C_i(\mathrm{IS}_m)$ represents the count of $\mathrm{IS}_m$ in node *i*. In what follows, system define a weight to quantify the computing load of $\mathrm{IS}_m$. Given an itemset IS*m*, the computing-load weight of $\mathrm{Is}_m$ over all the itemsets is defined as

$$\omega(IS_m) = \frac{C(IS_m)\times 2^m}{\sum_{j=2}^{M} C(IS_j)\times 2^j} \tag{1}$$

where 2*m* is the time complexity of decomposing *m*-itemset $C(\mathrm{IS}_m)$ is the count of $\mathrm{IS}_m$ over all available data nodes. Based on the definition of $\omega(\mathrm{IS}_m)$ in (1). System define the computing load of node *i*. Given a transaction database **D** partitioned over *p* nodes and a random itemset **X**, the computing load of node *i* is expressed as

$$W_i = \sum_{X \in IS} \omega(X) \times p_i(X) \tag{2}$$

The summation of all the computing-load over all nodes is one; thus, system have $\sum_{j=1}^{p} W_j = 1$.

A data distribution leads to high load-balancing performance if the weights *Wi* ($i \in [1, p]$) are identical. On the other hand, a large discrepancy of the *Wi* gives rise to poor load-balancing performance. System introduce the entropy measure as a load-balancing metric. Given a database **D** partitioned over *p* nodes, the load-balancing metric is expressed as

$$WB(D) = \frac{\sum_{j=1}^{p} W_j \log(W_j)}{\log(p)} \tag{3}$$

The WB*(D)* metric defined in the form of entropy has the following properties.

1) If WB*(D)* equals to 1 (i.e., WB*(D)* = 1*)*, decomposition load is perfectly balanced across all the nodes.

2) If WB*(D)* equals to 0 (i.e., WB*(D)* = 0*)*, decomposition load is concentrated on one node.

3) All the other cases are represented by 0 < WB(*D*) < 1.

## Optimize the performance

The aforementioned analysis confirms that if the length of itemsets to be decomposed is large, the decomposition cost will exponentially increase[7]. In this section, system conduct experiments to investigate the impact of dimensionality on Fidoop. Furthermore, in order to facilitate parallelism, Pfp groups frequent one- itemsets and distributes the data corresponding to these items to each computing node; such a grouping strategy is both time and space consuming. In this phase comparison is made within apriori and fp-growth algorithm which gives more detail difference of performance by both. In this comparison performance of fp-growth algorithm is best than apriori.

## Reduce performance time

Nevertheless, when the dimensionality approximately reaches 30, Fidoop's performance starts degrading. This is because the cost of decomposing a *k*-itemset is very expensive (i.e., 2*m*, *m* is determined by the dimensionality of the dataset). The increasing value of the *m* exponentially enlarges the running time of Fidoop.

## *Mathematical Model*

Let, System S is represented as: S = { B, L, T, H , J, K, F }

'S' be the | Parallel Mining of Frequent Itemsets Using Map Reduce the final set.

Identify the inputs as B, L, T

S = {B, L, T …

B= {B 1, B 2, B 3, B 4, …| 'B' given as user information.}

L= {L 1, L 2, L 3, L 4, …| 'L' given as user transaction dataset.}

T= {T 1, T 2, T 3, T 4, …| 'T' given as mining dataset.}
Identify the outputs as O
S = {B, L, T, H, J, K…

H= {H 1, H 2 …| 'H' is the Response as reduced transaction dataset.}

J= {J 1, J 2 …| 'J' is the Response as update database}

K= {K 1, K 2 …| 'K' is the Response show graphs }

Identify the functions as 'F'

S = {B, L, T, H, J, K, F…

F = { F1(), F2(), F3(), F4(), F5(),F6(),F7(),F8()}

F1(B)::Process Requests on login ,Registration.

F2(B)::Respond as Map reduce.}.

F3(L)::Process Requests on given load balancing.

F4 (L):: Respond as update database.

F5(T)::Process Requests on reducing processing time.

F6(T)::Respond as reducing dataset transaction.

## V. RESULT AND DISCUSSION

### A Dataset

In this work, system is using e-shopping mall data. All dataset is in text file format. This system will work on any mall dataset. In this system, algorithm on 5GB dataset is tested. Proposed algorithm is working efficient on 5GB high dimensional dataset. This system can handle large dataset(more than 10GB).

### B Results

This system is comparing Fidoop and Fidoop-hd on the basis of parallel mining of frequent itemset algorithm. In this system uses large dataset(5GB) and system analyses data well. MapReduce Framework is best way to handle this large dataset.

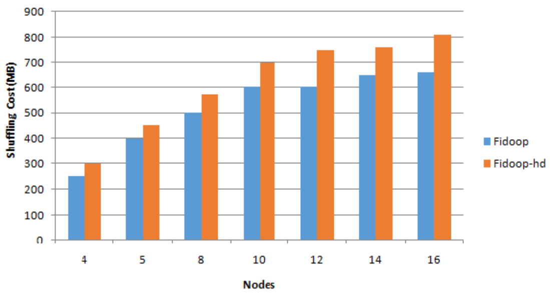


Fig.3. Shuffling cost for both algorithm

Comparison between fidoop and fidoop-hd is shown in fig.3 in terms of scalability. These are the expected results from system. In trems of scalability, results will be improved by 60%.

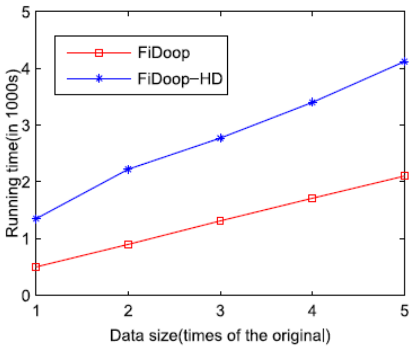


Fig.4. Running time for both algorithm

In fig.4 graph shows running time is improved than fidoop. Efficiency of this system is increased. Also system is tested for portability for different platforms like Windows, Linux version. In fig5 shows comparison of Apriori(sequential approach) and Fp-growth(Parallel approach). Fp-growth is efficient than Apriori.

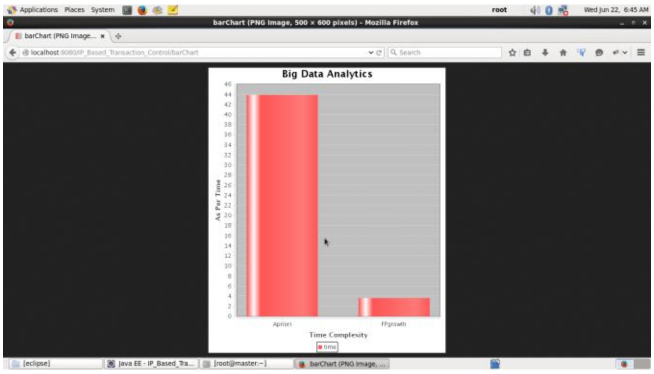


Fig.5. Comparison of Apriori and fp-growth Algorithm.

## VI. CONCLUSION

To solve the performance deterioration, load balancing and scalability challenges of sequential algorithm, various parallel algorithms were implemented. User gave an overview of such parallel algorithms. Unfortunately, in Apriori-like parallel FIM algorithms, each processor has to scan a database multiple times and to exchange an excessive number of candidate itemsets with other processors. Therefore, Apriori-like parallel FIM solutions suffer potential problems of high I/O and synchronization overhead, which makes it strenuous to scale up this parallel algorithms.The scalability problem, has been addressed by the implementation of a handful of FP-growth-like parallel FIM algorithms.

## VII. ACKNOWLEDGMENT

The authors would like to thank the researchers as well as publishers for making their resources available and teachers for their guidance. Finally, we would like to extend a heartfelt gratitude to friends and familymembers.

## REFERENCES

1. R. Agrawal and J. C. Shafer, ``Parallel mining of association rules´´, IEEE Trans. Knowl. Data Eng., vol. 8, no. 6, pp. 962969, Dec. 1996.

2. D. W. Cheung and Y. Xiao, ``Effect of data skewness in parallel mining of association rules'', Research and Development in Knowledge Discovery and Data Mining. Berlin, Germany: Springer,1998, pp. 4860.

3. M. J. Zaki,``Parallel and distributed association mining: A survey ´´, IEEE Concurrency, vol. 7, no. 4, pp. 1425,Oct./Dec. 1999

4. E.-H. Han, G. Karypis, and V. Kumar, ``Scalable parallel data mining for association rules´´, IEEE Trans. Knowl. Data Eng., vol. 12, no. 3,pp. 337352, May/Jun 2000 .

5. Pramudiono and M. Kitsuregawa, ``FP-tax: Tree structure based generalized association rule mining´´, Proc. 9th ACM SIGMOD, June 2004.

Workshop Res. Issues Data Min. Knowl. Disc., Paris,France, 2004, pp. 6063.

6. L. Liu, E. Li, Y. Zhang, and Z. Tang, ``Optimization of frequent itemset mining on multiplecore processor´´, Proc. 33rd Int. Conf. Very Large Data Bases, Vienna, Austria, 2007, pp. 12751285.

7. Chen et al., ``Tree partition based parallel frequent pattern miningon shared memory systems´´,Proc. 20th IEEE Int. Parallel Distrib.Process. Symp. (IPDPS), Rhodes Island, Greece, 2006, pp. 18.

8. J. Dean and S. Ghemawat, ``MapReduce: Simplified data processing on large clusters ´´,Commun. ACM, vol. 51, no. 1, pp. 107113, Jan. 2008.

9. J. Dean and S. Ghemawat, ``MapReduce: A flexible data processing tool´´,Commun. ACM,vol. 53, no. 1, pp. 7277, Jan. 2010.

10. M.-Y. Lin, P.-Y. Lee, and S.-C. Hsueh, ``Apriori-based frequent itemsetmining algorithms on MapReduce´´, Proc. 6th Int. Conf. Ubiquit. Inf.Manage. Commun. (ICUIMC), Danang, Vietnam, 2012, pp. 76:1 76:8.[Online].Available:http://doi.acm.org/10.1145/218475 1.2184842.

11. W. Lu, Y. Shen, S. Chen, and B. C. Ooi, ´´Efficient processing of k nearest neighbor joins using MapReduce´´, Proc. VLDB Endow., vol. 5,no. 10, pp. 10161027, 2012.

12. Yaling Xun, Jifu Zhang, and Xiao Qin, “FiDoop: Parallel Mining of Frequent Itemsets Using MapReduce” IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS: SYSTEMS, 2168-2216, 2015 IEEE.

13. Anjali Kadam, Nilam Patil,”Improving Performance ofParallel Mining of frequent Itemsets using Fidoop” Fifth Post Graduate Conference of Computer Engineering cPGCON-2016