TF-FIU Tree Based Parallel Mining Algorithm for Frequent Itemset Using Fi^{2}doop framework

S.Sivaranjani[1], R.Rajmohan[2], D.Dinagaran[3], P.Thirugnanam[4], MO.Ramkumar[5]
Department of Computer Science and Engineering[1, 2, 3, 4, 5]
IFET College of Engineering[1, 2, 3, 4, 5], Villupuram, India
skgr1994@gmail.com, rjmohan89@gmail.com, ddinagaran@gmail.com, thrgnanam@gmail.com

***Abstract*-Existing parallel mining algorithms for frequent itemsets lack a mechanism that enables automatic parallelization, load balancing, data distribution, and fault tolerance on large clusters. Frequent Itemsets Mining (FIM) is a core problem in association rule mining (ARM), sequence mining, and the like. Speeding up the process of FIM is critical and indispensable, because FIM consumption accounts for a significant portion of mining time due to its high computation and input/ output (I/O) intensity. Frequent itemsets mining algorithms can be divided into two categories, namely, Apriori and FP-growth schemes. Apriori is a classic algorithm using the generate-and-test process that generates a large number of candidate itemsets. Apriori has to repeatedly scan an entire database. Rather than considering Apriori and FP-growth, we incorporate the TF-FIUT algorithm in the design of our parallel FIM technique for reducing I/O overhead and parallelize the traditional FIUT algorithm for better performance. A pattern mining algorithm called Fi^{2}Doop is proposed by combining the TF-FIUT Algorithm and i^2 map reduce technique to optimize the performance of load balancing among the cluster nodes and avoid conditional pattern bases. Experiments using real world data demonstrate that our proposed solution is efficient and scalable.**

***Index Terms*-Frequent items ultrametric tree,Hadoop clusters,load balance,i^{2}MapReduce.**

I. INTRODUCTION

Frequent Itemsets Mining (FIM) [10] is a core problem in association rule mining (ARM), sequence mining, and the like. Speeding up the process of FIM is critical and indispensable, because FIM consumption accounts for a significant portion of mining time due to its high computation and input/ output (I/O) intensity. When datasets in modern data mining applications become excessively large, sequential FIM algorithms running on a single machine suffer from performance deterioration. To address this issue, we investigate how to perform FIM using MapReduce [9]—a widely adopted programming model for processing big datasets by exploiting the parallelism among computing nodes of a cluster. We show how to distribute a large dataset over the cluster to balance load across all cluster nodes, thereby optimizing the performance of parallel FIM.The main contributions of this paper are summarized as follows.

1) We made a complete overhaul toTF-FIUT (i.e., the TF based frequent items ultrametric trees method), and addressed the performance issues of parallelizing FIUT.
2) We adopted the parallel frequent itemsets mining method (i.e., Fi^{2}Doop) using the MapReduce programming model.
3) We proposed a data distribution scheme to balance load among computing nodes in a cluster.
4) We further optimized the performance of Fi^{2}Doop and reduced running time of processing high-dimensional datasets.
5) We conducted extensive experiments using a wide range of synthetic and real-world datasets, and we show that Fi^{2}Doop is efficient and scalable on Hadoop clusters.

II. RELATED WORK

A Tree Partitioning Method[1] is proposed for Memory Management in Association Rule MiningAll methods of association rule mining require the frequent sets of items, that occur together sufficiently often to be the basis of potentially interesting rules, to be first computed. The cost of this methodology increases in proportion to the database size, and also with its density.Densely-populated databases can give rise to very large numbers of candidates that must be counted.

Mining Association Rules [2]between Sets of Items in Large Databases is proposed. A large database of customer transactions is taken for analysis. Each transaction consists of items purchased by a customer in a visit. We present an efficient algorithm that generates all significant association rules between items in the database. The algorithm incorporates buffer management and novel estimation and pruning techniques. We also present results of applying this algorithm to sales data obtained from a large retailing company, which shows the effectiveness of the algorithm.

Fast Algorithms for mining association rules[3] consider the problem of discovering association rules between items in a large database of sales

transactions. They have also presented two new algorithms for solving this problem that are fundamentally different from the known algorithms. Empirical evaluation shows that these algorithms outperform the known algorithms by factors ranging from three for small problems to more than an order of magnitude for large problems and also show how the best features of the two proposed algorithms can be combined into a hybrid algorithm, called AprioriHybrid. Scale-up experiments show that AprioriHybrid scales linearly with the number of transactions. AprioriHybrid also has excellent scale-up properties with respect to the transaction size and the number of items in the database.

Algorithms for Computing Association Rules using a Partial-Support Tree [8]New algorithms for the extraction of association rules from binary databases. Most existing methods operate by generating "candidate" sets, representing combinations of attributes which may be associated, and then testing the database to establish the degree of association. This may involve multiple database passes, and is also likely to encounter problems when dealing with "dense" data due to the increase in the number of sets under consideration. Our method uses a single pass of the database to perform a partial computation of support for all sets encountered in the database, storing this in the form of a set enumeration tree. We describe algorithms for generating this tree and for using it to generate association rules.

An Efficient Algorithm of Frequent Itemsets Mining Based on MapReduce [4]Mainstream parallel algorithms for mining frequent itemsets (patterns) were designed by implementing FP-Growth or Apriori algorithms on MapReduce (MR) framework. Existing MR FP-Growth algorithms cannot distribute data equally among nodes, and MR Apriori algorithms utilize multiple map/reduce procedures and generate too many key-value pairs with value of 1; these disadvantages hinder their performance. This paper proposes an algorithm FIMMR: it firstly mines local frequent itemsets for each data chunk as candidates, applies prune strategies to the candidates, and then identifies global frequent itemsets from candidates. Experimental results show that the time efficiency of FIMMR outperforms PFP and SPC significantly; and under small minimum support threshold, FIMMR can achieve one order of magnitude improvement than the other two algorithms; meanwhile, the speedup of FIMMR is also satisfactory.

III. TF-FIU TREE BASED AND PARALLELIZE

In light of the MapReduce programming model, we design a parallel TF based frequent itemsets mining algorithm called Fi^{2}Doop. The design goal of Fi^{2}Doop is to build a mechanism that enables automatic parallelization, load balancing, and data distribution for parallel mining of frequent itemsets on large clusters.After the root is labeled as null, an itemset*P1, p2, . . . , pm* of frequent items is inserted as a path connected by edges *(p1, p2), (p2, p3), . . . , (pm−1, pm)*without repeating nodes, beginning with child *p*1 of the root and ending with leaf *pm* in the tree.

An FIU-tree is constructed by inserting all itemsets asits paths; each itemset contains the same number of frequentitems. Thus, all of the FIU-tree leaves are identical height. Each leaf in the FIU-tree is composed of two fields:named item-name and count. The count of an item-name is the number of transactions containing the itemset that is the sequence in a path ending with the itemname. A non-leaf node in the FIU-tree contains two fields: named item-name and node-link. A node-link is a pointer linking to child nodes in the FIU-tree T. The following algorithm is used for TF-FIU tree generation.

Algorithm TF-FIUT

1: function ALGORITHM 1(A): TF-FIUT(D, n)
2: h-itemsets = k-itemsets generation(D, MinSup);
3:for k = M down to [1]do
4: k-TF-FIU-tree = k-TF-FIU-tree generation (h itemsets);
5: frequent k-itemsets Lk = frequent k-itemsets generation (k-TF-FIUtree);
6:end for
7: end function
8: function ALGORITHM 1(B): K-FIU-TREE GENERATION ((h-itemsets))
9: Create the root of a k-TF-FIU-tree, and label it as null (temporary 0 th root)
10: for all ($k + 1 \leq h \leq M$) do
11: decompose each h-itemset into all possible k-itemsets, and union original k-itemsets;
12: for all (k-itemset) do
13: ⋯ build k-TF-FIU-tree( ); *here, pseudo code is omitted*;
14: end for
15: end for

IV. MAPREDUCE FRAME WORK

MapReduce [5] is a promising parallel and scalable programmingmodel for data-intensive applications and scientific analysis which is depicted in Fig 1. 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 Hadoop one of the most popular MapReduceimplementations isrunning on clusters where Hadoop distributed file system (HDFS) stores data to provide high aggregateI/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.

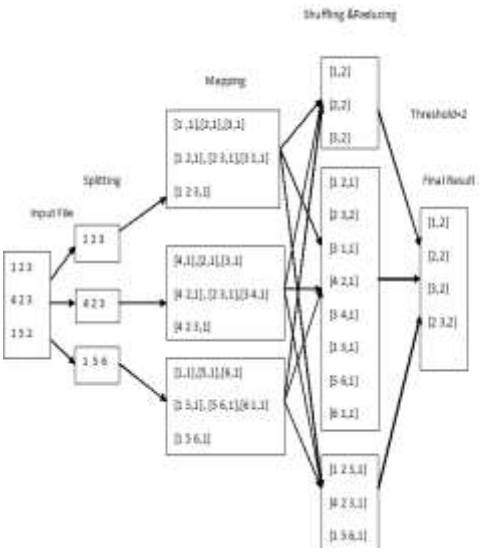

Fig.1 System Architecture

The *decompose*() function of the third MapReduce job accomplishes the decomposition process. If the length of an itemset is *m*, the time complexity of decomposing the itemset is *O(2m)*. Thus, the decomposition cost is exponentially proportional to the itemsets length step toward balancing load among data nodes of a Hadoop cluster is to quantitatively measure the total computing load of processing local itemsets and optimization approach to boost the speed of processing high-dimensional data. The flow of map reduce framework is depicted in fig 2.

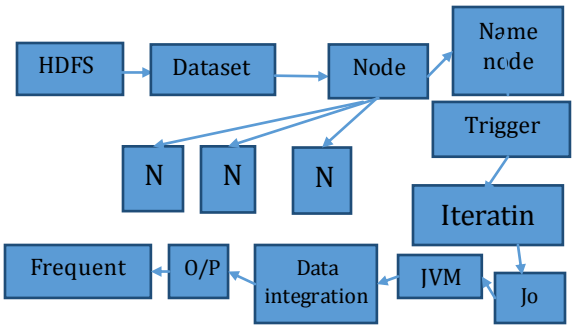

Fig.2 Flow diagram

V. MAP REDUCE BASED FI^{2}DOOP

I^{2}MapReduce [6], a novel incremental processing extension to MapReduce, the most widely used framework for mining bigdata is used in our model. We adopt the i2MapReduce, an extension to MapReduce that supports fine-grain incremental processing for both one- step and iterative computation as shown in fig 3. Fine-grain incremental processing is done using MRBG-Store. General-purpose iterative computation with modest extension is used to MapReduce API. Incremental processing for iterative computation is also adapted.

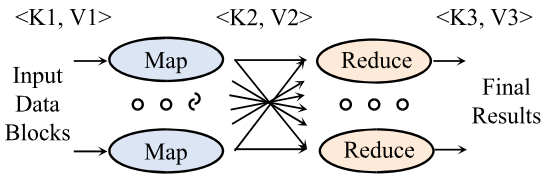

Fig.3 I^2 MapReduce

The design issues of Fi^{2}Doopbuilt on the MapReduce framework. Recall that the intermediate results provided by the mappers in the third MapReduce job areused to construct TF-FIU trees. The first MapReduce job is responsible for creating all frequent one-itemsets. A transaction database is partitioned intomultiple input files stored by the HDFS across data nodes of a Hadoop cluster. After performing the combination operation, each reducer emits key/value pairs, where the key is the number of each itemset and the value is each itemset and its count. The third MapReduce job is highly scalable, because the decomposition procedure of each mapper is independent of the other mappers. In other words, the multiple mappers can perform the decomposition process in parallel.

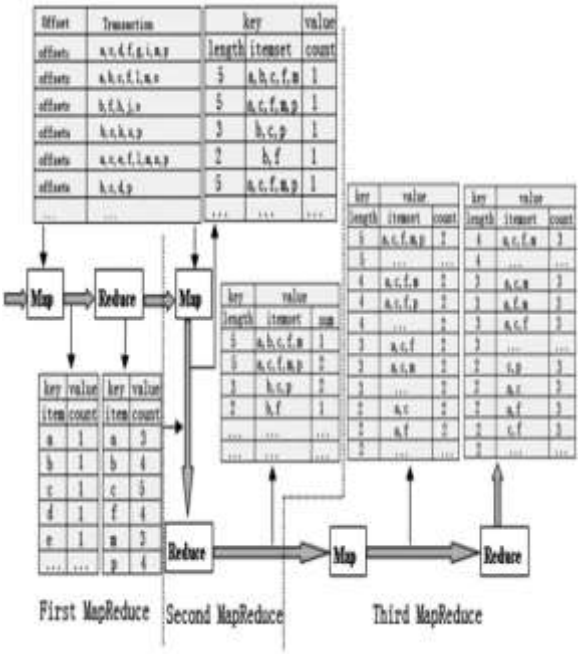

Fig.2 MapReduce based fi[2]Doop

VI. EXPERIMENTAL DESIGN

We evaluate the performance of Fi[2]Doop on our in-house Hadoop cluster equipped with 16 data nodes. Each node has an Intel Core i3 3.0 GHz processor, 4 GB MB main memory, and runs on the Ubuntu 15.10 operating system, on which Java JDK 8 and Hadoop 2.7.2 are installed. All the data nodes in the cluster have Gigabit Ethernet using networkinterface cards connected to Gigabit ports on the switch; the nodes can communicate with one another the secure shell protocol. We use the default Hadoop parameter configurations to set the replication factor and the number of Map and Reduce tasks.

To evaluate the performance of the proposed Fi[2]Doop, we use both synthetic and real-world datasets in our experiments.

1) *Synthetic Dataset:* We generated a series of synthetic datasets (i.e., the D1000W datasets) using the IBM quest market-basket synthetic data generator, which can be configured to create a wide range of datasets to meet the needs of various test requirements [7]. The number of items in each D1000W dataset is set to 1000 (represents the number of varieties of goods);
2) *Celestial Spectral Dataset:* We apply Fi[2]Doop to implement a parallel data mining application for celestial spectral data. We use the real-world celestial spectral dataset to evaluate speedup, load-balancing performance, as well as the impact of minimum support. The celestial spectral dataset used in our experiments has 5000000 transactions and 44 dimensions.

A. *MINIMUM SUPPORT*

Minimum support plays an important role in mining frequent itemsets. We increase minimum support thresholds from 0.001% to 0.004% with an increment of 0.0005%, thereby evaluating the impact of minimum support of our proposed algorithm using both celestial spectral and synthetic datasets.

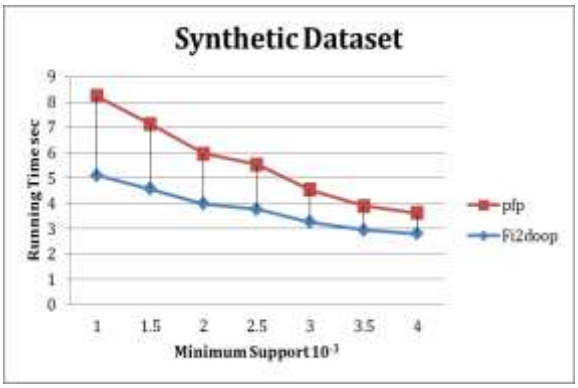

Fig.3(a) Synthetic Dataset

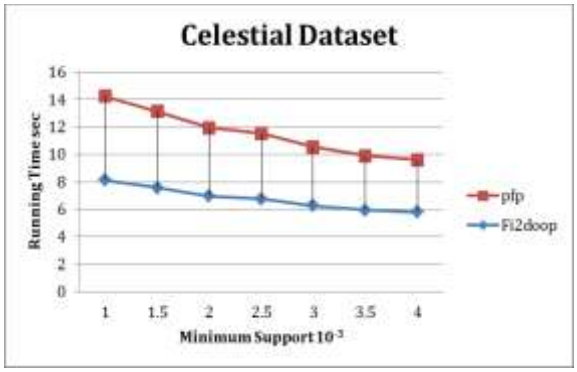

Fig.3(b) Celestial Dataset

Fig. 3(a)–(b) shows the execution times of the algorithms on synthetic datasets and celestial spectral dataset, respectively. It is evident from these experimental results that the performance of Fi[2]Doopis improved in the case of high-dimensional datasets.

B. *Load Balancing*

Fig.4 shows the impact of workload balance metric on running time measured in the unit of 100 sec.

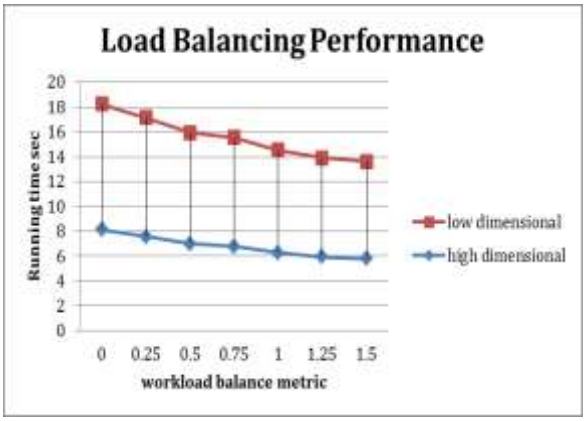

Fig 4 celestial spectral dataset

In this group of experiments, we measure Fi^{2}Doop workload balance metric on a low- and high-dimensional datasets. In the case of high-dimensional dataset, we test our algorithms using the celestial spectral dataset. Recall that our analysis shows that the load balancing mechanism of the third MapReduce job substantially improves the performance of Fi^{2}Doop.

C. Speedup

We evaluate the speedup performance of Fi^{2}Doop by increasing the number of data nodes in the test Hadoop cluster from 10 to 35 with an increment of 5. Fig. 5 reveals the speedup of the three schemes as a function of the number of data nodes in the Hadoop cluster.

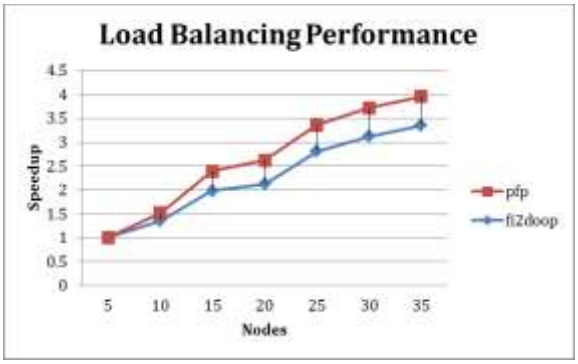

Fig.5 Load balancing performance of Fi^{2}doop

The experimental results illustrated in Fig5 shows that the reduced amount of itemsets being handled by each node, increases communication overhead between mappers and reducers.

D. Scalability

In this group of experiments, we evaluate the scalability of Fi^{2}Doop when the size of input dataset grows dramatically Fig. 6 clearly reveals that the overall execution time of Fi^{2}Doop goes up when the input data size is sharply enlarged.

Fig.6 Scalability of Fi^{2}Doop

VII. CONCLUSION

To solve the scalability and load balancing challenges in the existing parallel mining algorithms for frequent itemsets, we applied the MapReduce programming model to develop a parallel TF based frequent itemsets mining algorithm called Fi^{2}Doop. Fi^{2}Doop incorporates the TF based frequent items ultrametric tree or TF-FIU-tree rather than conventional FP trees, thereby achieving compressed storage and avoiding the necessity to build conditional pattern bases. Fi^{2}Doop seamlessly integrates three MapReduce jobs to accomplish parallel mining of frequent itemsets. We introduced a metric to measure the load balance of Fi^{2}Doop. As a future research direction, we will apply this metric to investigate advanced load balance strategies in the context of Fi^{2}Doop.

References

[1] S. Ahmed, F. Coenen, and P.H. Leng: A Tree Partitioning Method for Memory Management in Association Rule Mining. In Proc. of Data Warehousing and Knowledge Discovery, 6th International Conference (DaWaK 2004), Lecture Notes in Computer Science 3181, pp. 331–340, Springer-Verlag 2004.

[2] R. Agrawal, T. Imielinski, and A. Swami. Mining Association Rules between Sets of Items in Large Databases. In Proc. of the 1993 ACM SIGMOD Conference on Management of Data, Washington DC, pp. 207–216 1993.

[3] R. Agrawal and R. Srikant. Fast Algorithms for mining association rules. In Proc. VLDB'94, pp. 487–499 1994.

[4] K.Jayasri, R.Rajmohan, D.Dinagaran, "Analyzing the query performances of description logic based service matching using Hadoop",International Conference on Smart Technologies and Management for Computing, Communication, Controls, Energy and Materials (ICSTM), DOI:10.1109/ICSTM.2015.7225382, pp. 1-7, May, 2015.

[5] F. N. Afrati, J. D. Ullman, "Optimizing Multiway Joins in a Map-Reduce Environment", IEEE Transactions on Knowledge and Data Engineering, DOI: 10.1109/TKDE.2011.47, pp. 1282 – 1298, July, 2011.

[6] Yanfengzhang, Shimin Chen, Qiang Wang, "i2MapReduce: Incremental MapReduce for Mining Evolving Big Data", IEEE transaction on Knowledge and Data engineering, Vol. 27, No.7, DOI:10.1109/TKDE.2015.2397438, July 2015.

[7] L. Cristofor. (2001). Artool Project[J]. [Online]. Available:http://www.cs.umb.edu/laur/ARtool/, accessed Oct. 19, 2012.

[8] G. Goulbourne, F. Coenen, and P. Leng. Algorithms for Computing Association Rules using a Partial-Support Tree. *Journal of Knowledge-Based Systems* pp. 141–149, 13 2000.

[9] J.Dean and S.Ghemawat, "Mapreduce:Simplified data processing on large clusters", in Proc. 6th Conf. Symp. Opera. Syst. Des. Implementation, 2004, p.10.

[10] D.Chen et al., "Tree partition based parallel frequent pattern mining on shared memory systems", in Proc. 20th IEEE Int. Parallel Distribution Process. Symp. {1PDPS}, Rhodes Island Greece, 2006, pp. 1-8.

