FIDOOP-HD: Mining Frequent Datasets on Layer Clusters by Balanced Iterative Reducing and Clustering using Hierarchies

E.Shapna Rani [1#]
[1#]Assistant Professor, Department of Computer Science and Engineering,
Tagore Institute of Engineering and Technology, Salem, Tamil Nadu, India
[1#]shapnaedwin@gmail.com

T.Aarthi[2]
[2]Assistant Professor, Department of Computer Science and Engineering,
Tagore Institute of Engineering and Technology, Salem, Tamil Nadu, India
[2] aarthisivamani@gmail.com

Abstract— **Mining of data items plays a very crucial role in recent era as the World tends to manage with huge number of data's. Existing algorithm for frequent item set is Parallel Mining algorithm. But it lacks various mechanisms like Automatic Parallelization, Load Balancing, Data distribution and Fault Tolerance on large clusters. So a Parallel frequent item sets mining algorithm called FiDoop using the Balanced Iterative Reducing and Clustering using Hierarchies (BIRCH) is designed. To achieve compressed storage and avoid building conditional pattern bases, FiDoop is implemented on in-house Hadoop cluster and showed that FiDoop on the cluster is sensitive to data distribution and dimensions, because item sets with different lengths have different decomposition and construction costs. Also Birch performs faster, scans whole data only once, handles outlier better, superior to other algorithms in stability and scalability. To improve FiDoop's performance, workload balance metric is developed to measure load balance across the cluster's computing nodes. FiDoop-HD, an extension of Fi-Doop to speed up the mining performance for high-dimensional data analysis is developed extensive experiments using real-world celestial spectral data demonstrate that the proposed solution is efficient and scalable.**

Keywords— **Parallel Mining, Data distribution, Hadoop Clusters, BIRCH.**

I. INTRODUCTION

Frequent itemsets mining (FIM) is a core problem in association rule mining (ARM), sequence mining, and 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. To address this issue, FIM using MapReduce a widely adopted programming model for processing big datasets by exploiting the parallelism among computing nodes of a cluster is investigated.

Frequent itemsets mining algorithms can be divided into two categories, namely, *Apriori* and FP-growth schemes. Most previously developed parallel FIM algorithms were built upon the *Apriori* algorithm. Unfortunately, in *Apriori* like parallel FIM overhead, which make it strenuous to scale up these Parallel algorithms. Rather than considering *Apriori* and FP-growth, the frequent items ultrametric tree (FIU-tree) in the design of the parallel FIM technique is incorporated. To solve the aforementioned open problems, a parallel FIM algorithm called FiDoop using the MapReduce programming model is designed.

Then, data clustering is done by BIRCH, a hierarchical clustering method. Data clustering concerns how to group a set of objects based on their similarity of attributes and/or their proximity in the vector space. Balanced Iterative Reducing and Clustering using Hierarchies is an unsupervised data mining algorithm used to perform hierarchical clustering over particularly large data-sets.

II. METHODOLOGY

1. FIM is one of the most critical and time-consuming tasks in association rule mining (ARM). An often-used data mining task provides a strategic resource for decision support by extracting the most important frequent

patterns that simultaneously occur in a large transaction database.

2. Map Reduce is a popular data processing paradigm for efficient and fault tolerant workload distribution in large clusters. Two phases are available namely, Map phase and Reduce phase. The Map phase splits an input data into a large number of fragments.

3. Parallel FP-Growth (Pfp) is based on a popular FP-Growth algorithm. It is an open source machine learning library developed on Hadoop clusters. FP-Growth efficiently discovers frequent itemsets in which Constructing and mining a compressed data structure (i.e., FP-tree) rather than an entire database.

Birches do Hierarchical Clustering. BIRCH offers I/O cost linear in the size of the dataset. The BIRCH algorithm builds a clustering feature tree (CF tree) while scanning the data set. Each entry in the CF tree represents a cluster of objects and is characterized by a triple (N, LS, SS).

4. Clustering feature is that Given N d-dimensional data points in a cluster, X_i (i = 1, 2, 3, ... , N). CF vector of the cluster is defined as a triple CF = (N,LS,SS), N - number of data points in the cluster, LS - linear sum of the N data points and SS - square sum of the N data points.

III. PROPOSED WORK

The working structure of the proposed techniques is as follows

Complete overhaul to FIUT (i.e., the frequent items ultrametric trees method), and addressed the performance issues of parallelizing FIUT is made. The parallel frequent itemsets mining method (i.e., FiDoop) using the MapReduce programming model is proposed. Data distribution scheme to balance load among computing nodes in a cluster is also proposed. The performance of FiDoop and reduced running time of processing high-dimensional datasets is further optimized. Extensive experiments using a wide range of synthetic and real-world datasets, and we show that FiDoop is efficient and scalable on Hadoop clusters is conducted. BIRCH is local in that each clustering decision is made without scanning all data points and currently existing clusters. It exploits the observation that data space is not usually uniformly occupied and not every data point is equally important. It makes full use of available memory to derive the finest possible sub-clusters while minimizing I/O costs. It is also an incremental method that does not require the whole dataset in advance.

A. Mining Technique

In this procedure mining is divided into 4 Phases.

1) Frequent Itemset Mining

2) MapReduce Framework

3) Parallel FP-Growth

4) Birch Clustering

1) Frequent Itemset Mining: A parallel frequent itemsets mining algorithm called FiDoop is designed. 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.

The first phase of FIUT (frequent items ultrametric tree) 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 one-itemsets. The second MapReduce job scans the database again to generate *k*-itemsets by removing infrequent items in each transaction.

The second phase of FIUT involving the construction of a *k*-FIU tree and the discovery of frequent *k*-itemsets is handled by a third 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 two-itemsets. In the third MapReduce job, the generation of short itemsets is independent to that of long item-sets. In other words, long and short itemsets are created in parallel by our parallel algorithm.

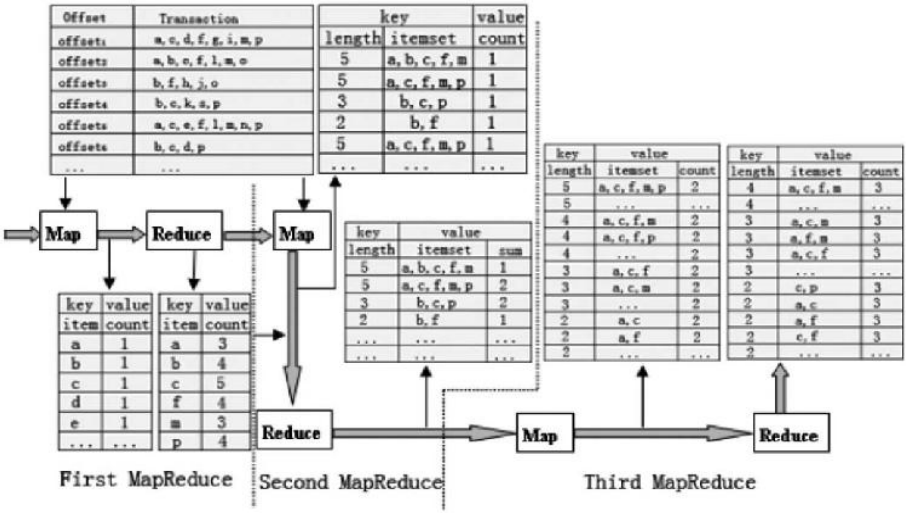

Figure 1: Map reducing of proposed work

2) *MapReduce Framework:* Recognizing that FiDoop exhibits high performance for low-dimensional databases, a dimensionality reduction scheme to efficiently handle high-dimensional data processing is designed and implemented. This approach called FiDoop-HD is an extension of the FiDoop algorithm. FiDoop-HD carries out the following steps to judiciously process high-dimensional data.

The output data of the second MapReduce job in FiDoop are, respectively, stored in multiple cache files according to itemset lengths. Thus, all k-itemsets are recorded in a file named k-file, whereas all $(k-1)$-itemsets are written to another file named $(k-1)$-file. FiDoop-HD decomposes the list of itemsets in a decreasing order of itemset length. After reading M-itemsets from a cache file, FiDoop-HD decomposes the M-itemsets into a list of $(M-1)$-itemsets. Note that, M is the maximal length of itemsets. Then, these itemsets combine original $(M-1)$-itemsets to be stored.

3) *Parallel FP-Growth:* PFP Growth is based on a popular FP-Growth algorithm. It is an open source machine learning library developed on Hadoop clusters. FP-Growth efficiently discovers frequent itemsets in which Constructing and mining a compressed data structure (i.e., FP-tree) rather than an entire database.

4) *Birch Clustering:* It has two phases. First phase scans the database to build an in-memory tree and the second phase is to apply clustering algorithm to cluster the leaf nodes. In describe, it has four phases.

- Phase 1: Scan all data and build an initial in-memory CF tree, using the given amount of memory and recycling space on disk.
- Phase 2: Condense into desirable length by building a smaller CF tree.
- Phase 3: Global clustering.
- Phase 4: Cluster refining – this is optional, and requires more passes over the data to refine the results.

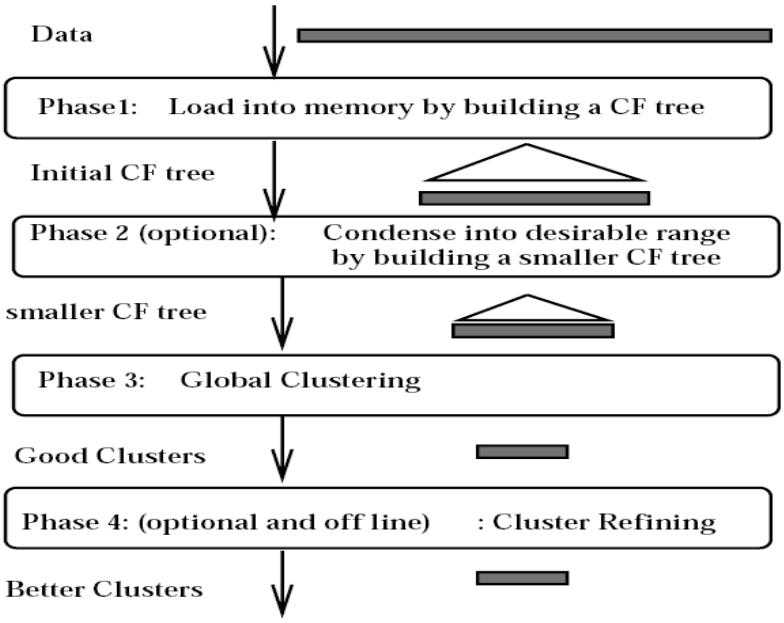

Figure 2: Phases of Birch algorithm

B. Clustering Feature(CF) Tree

In this proposed work, CF Tree is a very compact representation of the dataset because each entry in a leaf node is not a single data point but a subcluster. The tree size is a function of *T* (the larger the *T* is, the smaller the tree is). The Birch algorithm builds a dendrogram called clustering feature tree (CF tree) while scanning the data set. Each entry in the CF tree represents a cluster of objects and is characterized by a 3-tuple: (N, LS, SS), where N is the number of objects in the cluster and LS, SS are defined in the following

$$LS = \sum_{P_i \in N} \vec{P}_i \qquad (1)$$

$$SS = \sum_{P_i \in N} \left| \vec{P}_i \right|^2$$

CF entry is more compact, Stores significantly less than all of the data points in the sub-cluster. A CF entry has sufficient information to calculate D0-D4. Additivity theorem allows us to merge sub-clusters incrementally & consistently.

$$\mathbf{CF_1} + \mathbf{CF_2} = (N_1 + N_2, \vec{LS}_1 + \vec{LS}_2, SS_1 + SS_2) \qquad (2)$$

Each non-leaf node has at most B entries. Each leaf node has at most L CF entries, each of which satisfies threshold T. Node size is determined by dimensionality of data space and input parameter P (page size).

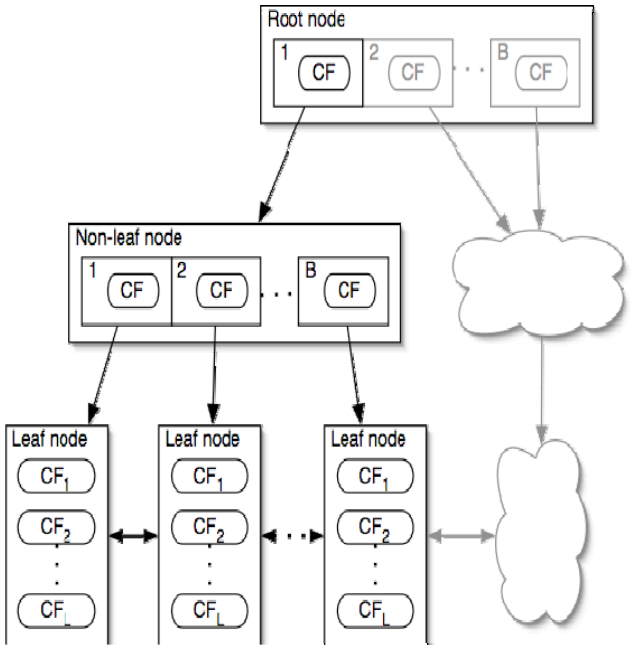

Figure 3: Clustering Feature (CF) Tree

(i) CF Tree Insertion

- Recurse down from root, find the appropriate leaf: Follow the "closest"-CF path, w.r.t. D0 / … / D4
- Modify the leaf: If the closest-CF leaf cannot absorb, make a new CF entry. If there is no room for new leaf, split the parent node
- Traverse back: Updating CFs on the path or splitting nodes.

(ii) CF Tree Rebuilding

- If we run out of space, increase threshold T. By increasing the threshold, CFs absorb more data
- Rebuilding "pushes" CFs over. The larger T allows different CFs to group together
- Reducibility theorem
  - Increasing T will result in a CF-tree smaller than the original
  - Rebuilding needs at most h extra pages of memory

Example of Birch

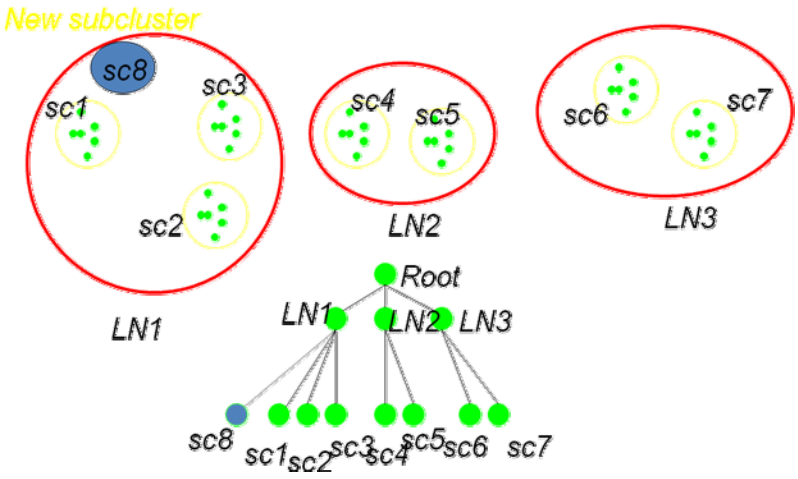

Figure 4: Actual Birch Cluster

Insertion Operation in BIRCH

If the branching factor of a leaf node cannot exceed 3, then LN1 is split.

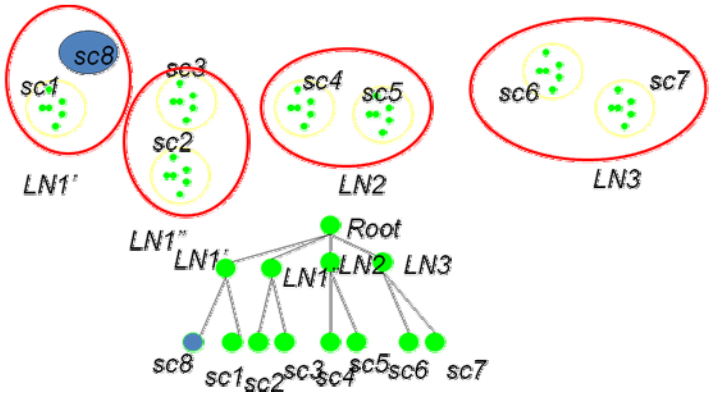

Figure 5: Birch Cluster after Insertion

If the branching factor of a non-leaf node cannot exceed 3, then the root is split and the height of the CF Tree increases by one.

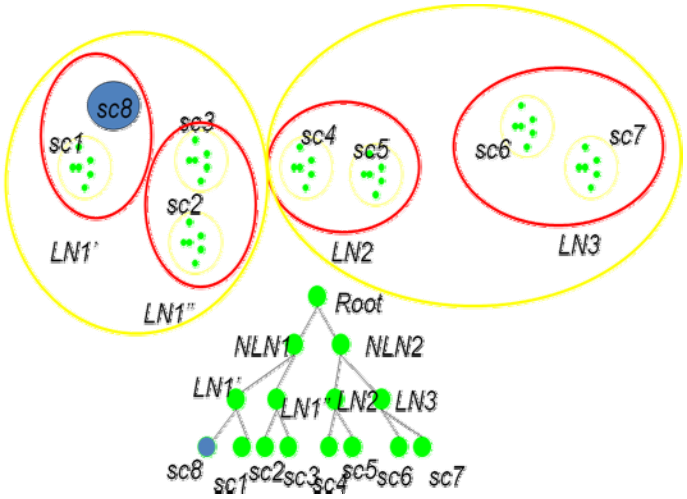

Figure 6: Root splits and height increase

IV: EXPERIMENTAL RESULTS

Input parameters

- Memory (M): 5% of data set
- Disk space (R): 20% of M
- Distance equation: D2
- Quality equation: weighted average diameter (D)
- Initial threshold (T): 0.0
- Page size (P): 1024 bytes

Our experimental results shows the comparison Birch clustering with K-means clustering

KMEANS clustering

DS	Time	D	# Scan	DS	Time	D	# Scan
1	43.9	2.09	289	1o	33.8	1.97	197
2	13.2	4.43	51	2o	12.7	4.20	29
3	32.9	3.66	187	3o	36.0	4.35	241

BIRCH clustering

DS	Time	D	# Scan	DS	Time	D	# Scan
1	11.5	1.87	2	1o	13.6	1.87	2
2	10.7	1.99	2	2o	12.1	1.99	2
3	11.4	3.95	2	3o	12.2	3.99	2

V. CONCLUSION

In this paper, to solve the scalability and load balancing challenges in the existing parallel mining algorithms for frequent itemsets, the MapReduce programming model is applied to develop a parallel frequent itemsets mining algorithm called FiDoop. FiDoop incorporates the frequent items ultrametric tree or FIU-tree rather than conventional FP trees, thereby achieving compressed storage and avoiding the necessity to build conditional pattern bases. FiDoop seamlessly integrates three MapReduce jobs to accomplish parallel mining of frequent itemsets. The third MapReduce job plays an important role in parallel mining; its mappers independently decompose item-sets whereas its reducers construct small ultrametric trees to be separately mined. The performance of FiDoop is improved by balancing I/O load across data nodes of a cluster.

FiDoop-HD, an extension of FiDoop is designed and implemented to efficiently handle high-dimensional data processing. FiDoop-HD decomposes the *M*-itemsets into balance of FiDoop. As a future research direction, proposed to apply this metric to investigate advanced load balance strategies in the context of FiDoop. For example, planning implement a data-aware load balancing scheme to substantially improve the load-balancing performance of FiDoop.

In this paper, a metric is introduced to measure the load of data placement scheme is conducive to balancing the amount of data stored in each heterogeneous node to achieve improved data processing performance. FiDoop is integrated with the data placement mechanism on heterogeneous clusters. One of the goals is to investigate the impact of heterogeneous data placement strategy on Hadoop-based parallel mining of frequent itemsets.

A CF tree is a height-balanced tree that stores the clustering features for a hierarchical clustering. Given a limited amount of main memory, BIRCH can minimize the time required for I/O. BIRCH is a scalable clustering algorithm with respect to the number of objects, and good quality of clustering of the data.

VI. FUTURE SCOPE

In addition to performance issues, energy savings and thermal management will be of our future research interests. We will propose various approaches to improving energy efficiency of Fidoop running on Hadoop clusters.

REFERENCES

[1] J. Zhang, X. Zhao, S. Zhang, S. Yin, and X. Qin, "Interrelation analysis of celestial spectra data using constrained frequent pattern trees," *Knowl.-Based Syst.*, vol. 41, pp. 77–88, Mar. 2013.
[2] M.-Y. Lin, P.-Y. Lee, and S.-C. Hsueh, "Apriori-based frequent itemset mining algorithms on MapReduce," in *Proc. 6th Int. Conf. Ubiquit. Inf. Manage. Commun. (ICUIMC)*, Danang, Vietnam, 2012, pp. 76:1–76:8.
[3] L. Cristofor. (2001). *Artool Project [J]*. [Online]. Available: http://www.cs.umb.edu/laur/ARtool/, accessed Oct. 19, 2012.
[4] 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. 1016–1027, 2012.
[5] J. Neerbek, "Message-driven FP-growth," in *Proc. WICSA/ECSA Compan. Vol.*, Helsinki, Finland, 2012, pp. 29–36.
[6] K.-M. Yu, J. Zhou, T.-P. Hong, and J.-L. Zhou, "A load-balanced distributed parallel mining algorithm," *Expert Syst. Appl.*, vol. 37, no. 3, pp 2459–2464, 2010.
[7] K. Yu and J. Zhou, "Parallel TID-based frequent pattern mining algorithm on a PC cluster and grid computing system," *Expert Syst. Appl.*, vol. 37, no. 3, pp. 2486–2494, 2010.