Parallel Mining of Frequent Itemsets Using MapReduce

[1]Raghavendra Naik, [2]Pramilarani K
[1]Student, [2]Senior Assistant Professor
[1]Department of Computer Science & Engineering,
[1]New Horizon College of Engineering Bangalore, India

Abstract: Existing parallel burrowing counts for visit itemsets don't have a part that engages modified parallelization, stack altering, data apportionment, and adjustment to non-basic disappointment on colossal clusters. As a response for this issue, we diagram a parallel visit itemsets mining estimation called FiDoop using the MapReduce programming model. To achieve pressed limit and go without building prohibitive case bases, FiDoop combines the normal things ultrametric tree, rather than common FP trees. In FiDoop, three MapReduce occupations are executed to complete the mining task. In the fundamental third MapReduce work, the mappers openly separate itemsets, the reducers perform blend errands by building little ultrametric trees, and the genuine mining of these trees autonomously. We realize FiDoop on our in-house Hadoop bundle. We exhibit that FiDoop on the gathering is sensitive to data allotment what's more, estimations, in light of the way that itemsets with different lengths have unmistakable rot and advancement costs. To gain ground FiDoop's execution, we develop a workload modify metric to measure stack change over the gathering's enrolling centers. We make FiDoop-HD, a development of FiDoop, to quicken the digging execution for high-dimensional data examination. Wide tests using genuine perfect unearthly data delineate that our proposed course of action is viable and flexible**.**

Index Terms **- Frequent Pattern Growth, Apriori, Rapid Association Rule Mining (RARM), ECLAT, Data Mining, Frequent Patterns, MapReduce.**

I. INTRODUCTION

Mining of frequent itemsets (FIM) is the main problem in mining of data using sequence mining algorithm, association rule of mining algorithm (ARM) and in the similar places. Among all types of data mining, frequent items pattern mining in the data mining subject. There are a lot of researches have been made and lots of efficient algorithms have been designed to search frequent pattern in the large transactional database. Agrawal et al for the first time in 1993, has proposed a concept market-based form of analysis of pattern for finding the relation between items that are fetched in a market places. The market-based analysis concept used the transactional databases and other databases and repositories which collects data in order to extract association rule's casual structures, their inter relations or frequent patterns among the dataset. Frequent patterns are the items or itemsets which repeatedly occur in database transactions with a user-specified frequency. An itemset whose occurrence frequency is greater than the minimum threshold will be considered as the frequent pattern. For example in market based analysis if the minimum threshold is 30% and bread appears with eggs and milk more than three times or at least three times then it will be a frequent itemset.

During the data mining of item pattern stage, there are different methods and techniques are used to get the candidate keys for frequently occurring patterns and generation of frequent patterns are carried out. In this stage, there are two main problems for mining the frequent pattern itemsets. The main problem is that the database is required to scan every time the search is doing, and the other one is each time when it scans the database, it generates a huge and complex dataset and it'll take huge time in scanning the same. These are the main two drawbacks in frequent pattern mining. There are a lot of studies have performed on this and efforts have been put to overcome and it results in finding different approaches and algorithms which are really useful. Some of the approaches are like Apriori, FP Growth, ASPMS RARM, ECLAT algorithms etc.

II. LITERATURE SURVEY

2.1. Mining of Frequent Itemsets

The Apriori algorithm is a classic way of mining frequent itemsets in a database. A selection of Apriori-like algorithms intention to shorten database scanning time by way of decreasing candidate itemsets. For instance, Park et al. proposed the direct hashing and pruning algorithm to govern the number of candidate two-itemsets and prune the database size using a hash technique. Inside the inverted hashing and pruning algorithm, every okay-itemset inside each transaction is hashed right into a hash desk. Berzal et al designed the tree-primarily based association rule algorithm, which employs a powerful statistics-tree structure to keep all itemsets to lessen the time required for scanning databases.

To enhance the execution of Apriori-like calculations, Han et al. proposed a novel approach called FP-development to abstain from creating an exorbitant number of applicant itemsets. The primary thought of FP-development is anticipating database into a reduced information structure, and after that utilizing the gap and-overcome technique to separate successive itemsets.

The main drawbacks of FP-development are: 1) The development of a large number of expandable FP trees in basic memory and 2) the repeated cross of Frequent pattern trees. To overcome this problem, Tsay et al. proposed another technique called FIUT, which

depends on visit things ultrametric trees to maintain a strategic distance from recursively crossing FP trees. Zhang et al. proposed an idea of obliged visit design trees to generously enhance the productivity of mining affiliation rules.

2.2. Parallel Mining of Frequent Itemsets

Mining algorithms for parallel frequent itemsets related to Apriori are classified into two groups, namely, count distribution, consists of count distribution and fast parallel mining, and parallel data mining (PDM)) and data distribution which consists of data and intelligent data distribution. In case of count distribution group, all candidate itemsets local support counts are calculated by individual processors of the parallel system. Then, all processors compute the total support counts of the candidates by exchanging the local support counts. The CD and PDM algorithms have simple communication patterns, because in every iteration each processor requires only one round of communication. In the data distribution camp, each processor only keeps the support counts of a subset of all candidates. Each processor is responsible for sending its local database partition to all the other processors to compute support counts. In general, DD has higher communication overhead than CD, because shipping transaction data demands more communication bandwidth than sending support counts.

The operation of the course in existing Apriori based mining accounts leads to an increase in indirect costs and synchronization. In order to reduce the time required for checking databases and transacting with assured items, parallel computations based on frequent pattern development were introduced as an alternative way to Apriori parallel computations. Pair of parallel calculations based on FP development was activated in parallel using multithreading on multi-core processors. A notable disadvantage of these parallel mining accounts is the futility of developing FP-based trees when the databases are large. This issue ends with brutal and multidimensional databases.

2.3. Parallel Data Mining on Clusters

Working groups and other multi-processor frameworks are not multiple configuration stages that address the underlying memory issue that has arisen in parallel mining of large-scale databases. The PFP-based parallel account limits the indirect costs of synchronization by dividing the FP tree and the list of successive components on processors. Tang and TORCIA benefited from the extended data bases of the units and K-prefix in checking the parallel area of the FIM budget. The new plan implementation is presented with frequent pattern trees.

III. FREQUENT DATA MINING

FI the MapReduce programming model, we design a different parallel dataset mining algorithm called FiDoop. FiDoop's design aims to build a mechanism that enables automatic parallelization, load balancing and data distribution for parallel mining of diverse materials on large groups. To facilitate FiDoop viewing, we summarized the coding used in this paper in Table I. In order to improve data storage efficiency and avoid building conditional rules, FiDoop incorporates the FIU tree concept instead of traditional FP trees. After creating h-itemsets in stage 1, an iterative process continues to run to create k-FIU trees and search for different k elements until a k value is reached from M to 2. In other words, the K-FIU trees are executed and k elements are detected repeatedly. Worse, it is not common to build a K-FIU tree, the most important and time-consuming phase. The K-FIU tree algorithm (h-itemsets) constructs the K-FIU tree by analyzing all elements h to k elements, where k + 1 h h M. M. Then, the original k-itemsets are calculated to build the K-FIU tree. The generation of k-decomposing elements requires all h - itemsets (h> k); thus, decomposition is performed sequentially from long to short elements. As such, we improve the serial FIUT algorithm as follows.

1) The main period of FIUT including two rounds of examining a database is executed as two MapReduce occupations. The primary MapReduce work is in charge of the first round of checking to make visit one itemsets. The second MapReduce work filters the database again to produce k-itemsets by expelling occasional things in every exchange.

2) The second period of FIUT including the development of a k-FIU tree and the revelation of incessant k-itemsets is taken care of by a third MapReduce work, in which h-itemsets ($2 \leqslant h \leqslant M$) are straightforwardly disintegrated into a rundown of $(h-1)$- itemsets, $(h-2)$- itemsets, . . . , and two itemsets. In the third MapReduce work, the age of short itemsets is free to that of long itemsets. At the end of the day, long and short itemsets are made in parallel by our parallel calculation. Such an advancement approach tackles the parallelization issue of Algorithms.

The three MapReduce tasks of our proposed Fi-Doop are clarified in detail.

The primary MapReduce work finds every single incessant thing or regular one-itemsets. In this stage, the contribution of Map undertakings is a database, and the yield of Reduce errands is all regular one-itemsets. The second MapReduce work filters the database to produce k-itemsets by evacuating occasional things in every exchange. The last MapReduce work—the most entangled one of the three—builds k-FIU-tree and mines all regular k-itemsets.

To encourage the portrayal of FiDoop, we audit the affiliation govern and condense the FIUT calculation's fundamental thought and we present the MapReduce programming structure.

Association Rules

ARM gives a key asset to choice help by extricating the most essential incessant examples that at the same time happen in a huge exchange database. A common ARM application is advertise crate investigation. An affiliation lead, for instance, can be "if a client purchases An and B, at that point 90% of them likewise purchase C." In this illustration, 90% is the certainty of the run the show.

Aside from certainty, bolster is another measure of affiliation administers, every one of which is a suggestion as $X \Rightarrow Y$. Here, X and Y are two itemsets, and $X \cap Y = \emptyset$. The certainty of an administer $X \Rightarrow Y$ is characterized as a proportion between support($X \cup Y$) and support(X). Note that, an itemset X has bolster s if s% of exchanges contains the itemset. We signify s = support(X); the help of the manage $X \Rightarrow Y$ is support($X \cup Y$).

A definitive goal of ARM is to find all decides that fulfill a client indicated least help and least certainty.

The ARM procedure can be deteriorated into two stages: 1) recognizing all regular itemsets whose help is more prominent than the base help and 2) shaping restrictive ramifications rules among the continuous itemsets. The main stage is more testing and muddled than the second one. Accordingly, most earlier investigations are essentially centered around the issue of finding continuous itemsets.

FIUT

The FIUT approach embraces the FIU-tree to upgrade the effectiveness of mining regular itemsets. FIU-tree is a tree structure developed as takes after.

1) After the root is marked as invalid, an itemset p1, p2, . . . , pm of incessant things is embedded as a way associated by edges (p1, p2), (p2, p3), . . . , (pm−1, pm) without rehashing hubs, starting with tyke p1 of the root and consummation with leaf pm in the tree.

2) A FIU-tree is developed by embeddings all itemsets as its ways, each itemset contains a similar number of continuous things. Along these lines, the greater part of the FIU-tree leaves are indistinguishable tallness.

3) Each leaf in the FIU-tree is made out of two fields: named thing name and check. The tally of a thing name is the quantity of exchanges containing the itemset that is the arrangement in a way finishing with the thing name.

Non-leaf hubs in the FIU-tree contain two fields: named thing name and hub connect. A hub connect is a pointer connecting to youngster hubs in the FIU-tree. The FIUT calculation comprises of two key stages. The principal stage includes two rounds of checking a database. The principal check produces visit one-itemsets by registering the help of all things, while the second sweep brings about k-itemsets by pruning every single rare item in every exchange record. Note that k indicates the amount of objects continuing in the exchange. In stage two, a k-FIU-tree is over and again built by disintegrating every h-itemset into k-itemsets, where $k + 1 \leqslant h \leqslant$ (M is the maximal estimation of k), and union-ing unique k-itemsets. At this stage, the second phase begins with the extraction of all successive materials in the light of K-FIU tree leaves without crossing the tree repeatedly. Contrasted and the FP-development technique, FIUT altogether lessens the processing time and storage room by deflecting overhead of recursively seeking and navigating restrictive FP trees.

MapReduce Framework

MapReduce is a programming model of parallel and adaptable operation for applications and logical investigation. A MapReduce program communicates an extensive appropriated calculation as a grouping of parallel tasks on datasets of key/esteem sets. A MapReduce operation has two stages, namely the Map and Reduce stage. The Map stage parts the info information into countless, which are equitably dispersed to Map assignments over the hubs of a group to process. Each Map errand takes in a key-esteem combine and afterward creates an arrangement of middle of the road key-esteem sets.

After the MapReduce runtime framework gatherings and sorts all the middle qualities related with a similar halfway key, the runtime framework conveys the transitional qualities to Reduce errand.

Table 1 Symbol and Annotation

Symbol	Annotation
minsupport	User-specified Minimum Support threshold
k-itemsets	Itemsets containing k items
k-FIU-tree	FIU tree constructed by all k-itemsets
M	The maximum value of k
IS_m	Itemsets in which the length of each itemset is m

Algorithm:FIUT
1.**function** ALGORITHM 1(A): FIUT(D, n)

```
2:   h-itemsets = k-itemsets generation(D, MinSup);
3:   for k = M down to 2 do
4:       k-FIU-tree = k-FIU-tree generation (h-itemsets);
5:       frequent k-itemsets Lk = frequent k-itemsets generation (k-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-FIU-tree, and label it as null (temporary 0th root)
10:   for all (k + 1 ⩽ h ⩽ 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-FIU-tree( ); here, pseudo code is omitted;
14:      end for
15:   end for
16: end function
```

Algorithm: FiDoop-HD-MiningMap: High-Dimensional Optimization for Map() Function

```
Input: k-file /*k-file(2 ⩽ k ⩽ M) is used to store the frequent k-itemsets generated in the second MapReduce.*/
Output: (k-1)-FIU-tree
1: function MAP(key k, values k-file)
2:   for all (k is from M to 2) do
3:     for all (k-itemset in k-file) do
4:       decompose(k-itemset, k-1, (k-1)-itemsets);
          /*Each k-itemset is only decomposed into (k-1)-itemsets */
5:       (k-1)-file ← the decomposed (k-1)-itemsets union the original (k-1)-itemsets in
         (k-1)-file;
6:       for all (t-itemset in (k-1)-file) do
7:         t − FIU − tree ← t-FIU-tree generation(local-FIU-tree, t-itemset);
8:         output(t, t-FIU-tree);
9:       end for
10:    end for
11:  end for
12: end function
```

As we know that the performance of FiDoop algorithm is very high at low dimensional databases, we need to implement a new model or algorithm which performs well in the case of high dimensional data processing. This method is called FiDoop-HD which is an extended version of traditional FiDoop algorithm. The proposed fidoop-HD performs some steps over high-dimensional data processing. 1) 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. 2) The elements in the list of items are decomposed by Didoop-HD based on their descending of length. After reading M-itemsets from a cache file, FiDoop-HD decomposes the M-itemsets into a list of (M − 1)-itemsets.

IV. SYSTEM DESIGNS

4.1 Usecase Diagram And Activity Diagram

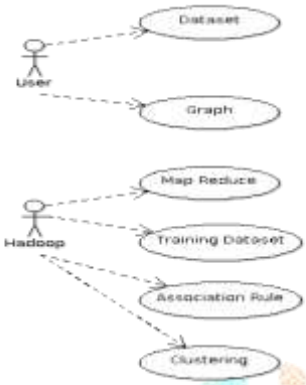


Fig. 4.1 a)Use-case diagram **Fig 4.1 b) Activity Diagram**

V. PROPOSED SYSTEM OVERVIEW

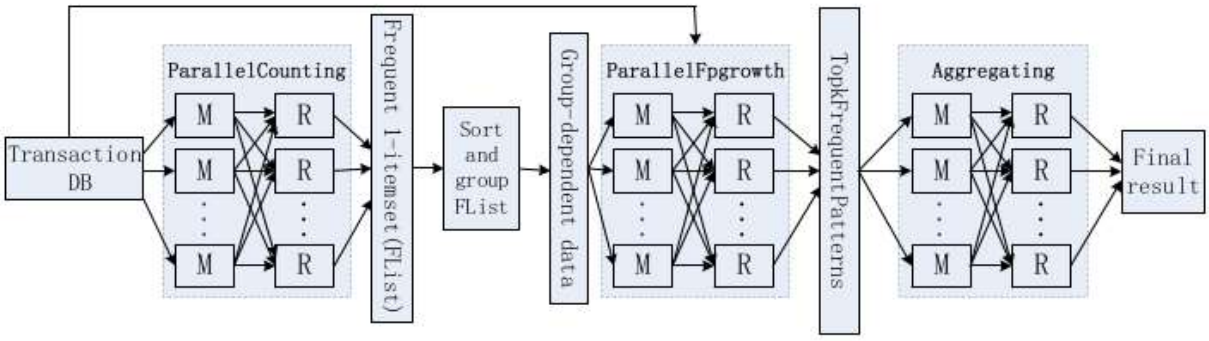


Fig.5.1 Architecture of Frequent itemset mapping

In the initial MapReduce job, every plotter consecutive reads every dealings from its native input split on an information node to get native 1-itemsets. Next, world 1-itemsets square measure made by a particular reducer that merges native 1- itemsets sharing an equivalent key (i.e., item name). The output of those reducers includes the worldwide frequent 1-itemsets alongside their counts. The second step kinds these world frequent 1-itemsets during a decreasing order of frequency; the sorted frequent 1-itemsets square measure saved during a cache named FList, that becomes Associate in nursing input of the second MapReduce job in FiDoop-DP.

The second MapReduce job applies a second-round scanning on the info to repartition info to create a whole dataset for item teams within the map section. Every reducer conducts native FP-Growth supported the partitions to get all frequent patterns.

The last MapReduce job aggregates the second MapReduce job's output (i.e., all the frequent patterns) to generate the final frequent patterns for each item. For example, the output of the second MapReduce job includes three frequent patterns, namely, 'abc', 'adc', and 'bdc'. Using these three frequent patterns as an input, the third MapReduce job creates the final results for each item as 'a: abc,adc', 'b: abc,bdc', 'c: abc,adc,bdc', and 'd: adc,bdc'.

An overview system consists of the following steps.

In lightweight of the MapReduce programming model, we tend to style a parallel frequent itemsets mining rule known as FiDoop. the planning goal of FiDoop is to make a mechanism that allows automatic parallelization, load equalization, and information distribution for parallel mining of frequent itemsets on massive clusters.

Aiming to improve information storage potency and to avert building conditional pattern bases, FiDoop incorporates the idea of FIU-tree instead of ancient FP trees. we tend to observe that parallelizing the serial rule FIUT may be a challenge [see conjointly the

most program of the FIUT rule in rule 1(A)]. Once h-itemsets area unit generated in section one, associate unvarying method is repeatedly running to construct k-FIU trees and to get frequent k-itemsets till the k price is from M to a pair of. In alternative words, the development of k-FIU trees and therefore the discovery of frequent k-itemsets area unit dead during a consecutive manner. Even worse, it's nontrivial to construct k-FIU-tree, that is that the most vital and time intense section. The k-FIU-tree-generation rule (h-itemsets) [see rule 1(B)] constructs a k-FIU tree by rotten every h-itemset into k-itemsets, wherever k + one ⩽ h ⩽ M. Then, the union of original k-itemsets is calculated to construct the k-FIU tree. The generation of the k-itemsets needs rotten all doable h-itemsets (h > k); so, the decomposition method is consecutive performed ranging from long to short itemsets. As such, we tend to improve the serial FIUT rule as follows.

1) The primary section of FIUT involving 2 rounds of scanning an info is enforced within the style of 2 MapReduce jobs. The primary MapReduce job is accountable for the primary spherical of scanning to make frequent one itemsets. The second MapReduce job scans the info once more to come up with k-itemsets by removing sporadic things in every dealing

2) The second section of FIUT involving the development of a k-FIU tree and also the discovery of frequent k-itemsets is handled by a 3rd MapReduce job, during which h-itemsets (2 ⩽ h ⩽ M) ar directly rotten into an inventory of (h − 1)-itemsets, (h − 2)-itemsets, . . . , and two- itemsets.

4.2 FiDoop-HD

Recognizing that FiDoop exhibits high performance for low-dimensional databases, we have a tendency to style and implement a spatial property reduction theme to expeditiously handle high-dimensional processing. This approach referred to as FiDoop-HD is associate degree extension of the FiDoop algorithmic program. FiDoop-HD carries out the subsequent steps to judiciously method high-dimensional knowledge.

1) The output knowledge of the second MapReduce job in FiDoop ar, severally, keep in multiple cache files in step with itemset lengths. Thus, all k-itemsets ar recorded in a very file named k-file, whereas all (k−1)-itemsets ar written to a different file named (k − 1)-file.

2) FiDoop-HD decomposes the list of itemsets in a very decreasing order of itemset length. When reading M-itemsets from a cache file, FiDoop-HD decomposes the M-itemsets into a listing of (M − 1)-itemsets. Note that, M is that the greatest length of itemsets. Then, these itemsets mix original (M − 1)-itemsets to be keep. Next, FiDoop-HD hundreds a cache file to store (M − 1)-itemsets, that ar rotten into (M − 2)-itemsets to be unioned into the first (M − 2)-itemsets. This procedure is repeatedly applied till the complete decomposition method is accomplished.

The mappers of the third MapReduce job solely decompose k-itemset into (k − 1)-itemsets instead of into two-itemsets (see algorithmic program 6). Just in case of multiple files keep on a knowledge node, the node consecutive hundreds and processes the files. In another algorithmic program, the price of mouldering Associate in Nursing m-itemset into (m−1)-itemsets is shapely as cm−1 m. Given a file storing all itemsets whose length is m, the decomposition value of the file is C(ISm) ×cm−1 m, wherever C(ISm) is that the count of belief within the file. Hence, the time complexness of the whole method is approximated as max(C (ISi)) × (cM−1 M +cM−2 M−1 +・・・+c23), which might be any written as max(C (ISi)) × (M × (M + 1)/2), two < i ⩽ M.

The time complexness of FiDoop-HD is far not up to that of FiDoop. Such a performance improvement is clear from the experimental results. We have a tendency to conduct the measurability study victimisation datasets that containing ten 000 000 transactions, during which the typical group action size varies anyplace from twenty to one hundred in step twenty. As an example, reveals that our answer well improves the performance of FiDoop. We have a tendency to implement FiDoop on a Hadoop cluster, wherever large amounts of information ar managed by HDFS. It's essential and demanding to handle the I/O performance problems in FiDoop-HD thanks to the subsequent reasons. First, itemsets rotten within the previous stages ought to be saved in new files for later phases. Second, FiDoop-HD will inherently incorporate a load-balancing policy, as a result of every node processes the files storing itemsets with a standardized length.

5.1 Modules

1. Data Partitioning
2. MapReduce Job
3. Parallel FP-Growth

5.1.1Data Partitioning

Data partition transactions by considering correlations among transactions and things before the parallel mining method. That is, transactions with a good similarity are divided into one partition so as to stop the transactions from being repeatedly transmitted to remote nodes. We have a tendency to adopt the Voronoi diagram-based knowledge partitioning technique that is contributive to maintaining knowledge proximity, particularly for multi-dimensional knowledge. Therefore, once the second MapReduce job is launched, a brand new Voronoi diagram based mostly knowledge partitioning strategy is deployed to reduce unessential redundant group action transmissions.

5.1.2 MapReduce Job

MapReduce may be a promising parallel and scalable programming model for data-intensive applications and scientific analysis. A MapReduce program expresses an outsized distributed computation as a sequence of parallel operations on datasets of key/value pairs.

A MapReduce computation has 2 phases, namely, the Map and scale back phases. The Map section splits the computer file into an outsized range of fragments, that area unit equally distributed to Map tasks across the nodes of a cluster to method. Every Map task takes during a key-value try and so generates a group of intermediate key-value pairs

V. PROPOSED SYSTEM FEATURE AND SAMPLE OUTPUT

In this paper, we introduced a metric to measure the load balance of FiDoop. As a future research direction, we will apply this metric to investigate advanced load balance strategies in the context of FiDoop. For example, we plan to implement a data-aware load balancing scheme to substantially improve the load-balancing performance of FiDoop. In one of our previous studies, we have addressed the data-placement issue in heterogeneous Hadoop clusters, where data are placed across nodes in a way that each node has a balanced data processing load.

Advantages

Advantageous to do verifications and forecasts of prediction patterns timely the way to remove patterns from the uninterrupted time data stream is, splitting the original decision, converting the subsequence into a kind of symbol data such as view pattern of featured space point), then classifying such symbol data, producing similar product or product collection the major different is how to sales the items pruning algorithm.

Fig 9.1 Main page of the application

Fig 9.2 Adding Hadoop nodes

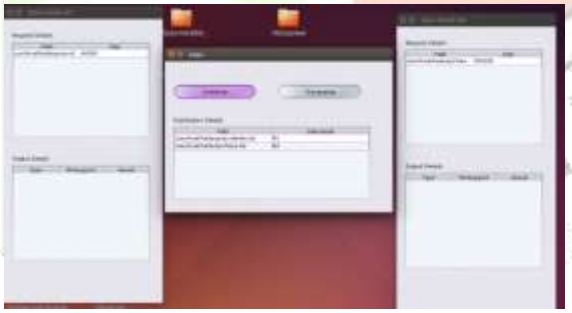
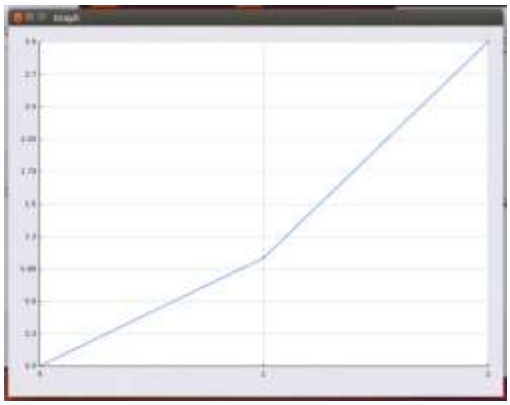


Fig 9.3 Data distribution across the nodes



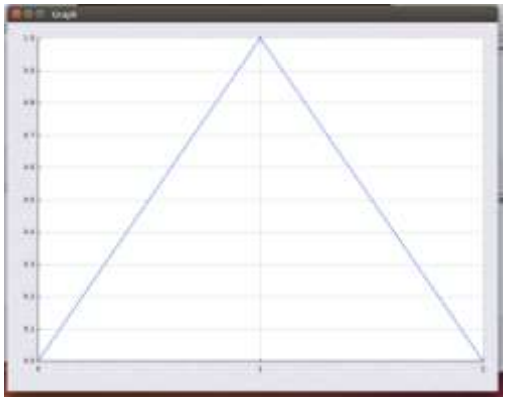


Fig 9.4 Data distribution across the nodes graph

VI. CONCLUSION

To solve the scalability and load balancing challenges in the existing parallel mining algorithms for frequent item sets, we applied the Map Reduce programming model to develop a parallel frequent item sets mining algorithm called FiDoop. FiDoop incorporates the frequent items ultra metric 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 Map Reduce jobs to accomplish parallel mining of frequent item sets. The third Map Reduce job plays an important role in parallel mining.

REFERENCES

[1] Sourav S. Bhowmick Qiankun Zhao, "Association Rule Mining: A Survey," Nanyang Technological University, Singapore, 2003.

[2] Yaling Xun, Jifu Zhang, and Xiao Qin, "FiDoop: Parallel Mining of Frequent Itemsets Using MapReduce", IEEE Transactions on Systems, Man, and Cybernetics: Systems, Vol. 46, No. 3, March 2016.

[3] Shamila Nasreen, Muhammad Awais Azam, Khurram Shehzad, Usman Naeem, Mustansar Ali Ghazanfar, "Frequent Pattern Mining Algorithms for Finding Associated Frequent Patterns for Data Streams: A Survey," The 5th International Conference on Emerging Ubiquitous Systems and Pervasive Networks, EUSPN-2014.

[4] J.R.Jeba, Dr.S.P.Victor, "Comparison of Frequent Item Set Mining Algorithms", International Journal of Computer Science and Information Technologies, Vol. 2 (6) , 2011.

[5] Jiawei Han • Hong Cheng • Dong Xin • Xifeng Yan, "Frequent pattern mining: current status and future Directions, "Data Mining Knowl Discov, vol. 15, no. I, p. 32, 2007.

[6] Iqbal Gondal and Joarder Kamruzzaman Md. Mamunur Rashid, "Mining Associated Sensor Pattern for data stream of wireless networks," in PM2HW2N '13, Spain, 2013, p. 8.

[7] "Data Mining Algorithms In R/Frequent Pattern Mining/The FP-Growth Algorithm" Wikibooks, open books for an open world.

[8] Seema Tribhuvan, Bharti. P. Vasgi, "Parallel Frequent Itemset Mining for Big Datasets using Hadoop-MapReduce Paradigm", International Journal of Advanced Research in Computer and Communication Engineering (ISO 3297:2007) Certified Vol. 6, Issue 6, June 2017

[9] M.A. Azam, Loo J., Naeem, Usman and Khan, S.K.A. and Lasebae, A. and Gemikonakli Azam, "A Framework to Recognise Daily Life Activities with Wireless Proximity and Object Usage Data", In 3rd IEEE International Symposium on Personal, Indoor and Mobile Radio Communication 2012, Sydney, Australia, 2012, p.6.

[10] Imielienskin T. and Swami A. Agrawal R., "Mining Association Rules Between set of items in largedatabases", in Management of Data, 1993, p. 9.

[11] M. H. Dunham, Y. Xiao, L. Gruenwald and Z. Hossain, "A Survey of Association Rules," http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.91.1602.

[12] M. Chen, and P.S. Yu J.S. Park, "An Effective Hash Based Algorithm for Mining Association Rules" in ACM SIGMOD Int'l Conf. Management of Data, May, 1995.

[13] Yahoo! Hadoop Tutorial, http://developer.yahoo.com/hadoop/tutorial/index.html

[14] S. Ghemawat, H. Gobioff and S. Leung, "The Google File System", in ACM SIGOPS Operating Systems Review, vol. 37, no. 5, pp. 29-43, 2003.

[15] K-H. Lee, Y-J. Lee, H. Choi, Y. D. Chung and B. Moon, "Parallel Data Processing with MapReduce: A Survey", in ACM SIGMOD Record, vol. 40, no. 4, pp. 11-20, 2011.

[16] F. Kovacs and J. Illes, "Frequent Itemset Mining on Hadoop" in Proceedings IEEE 9th International Conference on Computational Cybernetics (ICCC), Hungry, 2013, pp. 241-245.

[17] M-Y. Lin, P-Y. Lee and S-C. Hsueh, "Apriori-based Frequent Itemset Mining Algorithms on MapReduce" in Proceedings 6th International Conference on Ubiquitous Information Management and Communication (ICUIMC '12), ACM, New York, 2012, Article 76.

[18] S. Moens, E. Aksehirli and B. Goethals, "Frequent Itemset Mining for Big Data" in Proceedings IEEE International Conference on Big Data, 2013, pp. 111-118.

[19] J. Dean, and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters" in ACM Commun., vol. 51, pp. 107-113, 2008.

[20] K-H. Lee, Y-J. Lee, H. Choi, Y. D. Chung and B. Moon, "Parallel Data Processing with MapReduce: A Survey" in ACM SIGMOD Record, vol. 40, no. 4, pp. 11-20, 2011.

[21] X. Y. Yang, Z. Liu and Y. Fu, "MapReduce as a Programming Model for Association Rules Algorithm on Hadoop" in Proceedings 3rd International Conference on Information Sciences and Interaction Sciences (ICIS), 2010, vol. 99, no. 102, pp. 23-25.