ISSN(Online) : 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

An ISO 3297: 2007 Certified Organization *Vol.5, Special Issue 5, June 2017*

8th One Day National Conference on Innovation and Research in Information Technology (IRIT- 2017)

Organized by

Departments of ISE, CSE & MCA, New Horizon College of Engineering, Bengaluru, Karnataka 560103, India

FiDoop-DP: Data partitioning in frequent itemset mining on HaDoop clusters.

Gagana Vijayavarshini[1], Krithika V.Rao[1], M S Shobha[2]

B. E Student, Dept. of ISE, New Horizon College of Engineering, Bangalore, Karnataka, India [1]

Senior Assistant Professor, Dept. of ISE, New Horizon College of Engineering, Bangalore, Karnataka, India [2]

ABSTRACT: Parallel datamining involves the study and definition of parallel algorithms, methods and tools for the extraction of useful data from massive data using high-performance architectures. Frequent itemsets mining focuses on looking at sequences of actions. The already available parallel algorithms for frequent itemsets lack mechanism of automatic parallelization, load balancing, data distribution and fault tolerance on large clusters. We design FIDOOP using the MapReduce Programming model as a solution to this problem. FIDOOP is a frequent itemset mining algorithm which incorporates the frequent itemsets ultrametric tree. Which in turn helps to achieve compressed storage and avoids building conditional pattern bases . FIDOOP consists of three MapReduce jobs to accomplish mining task. The third MapReduce job is the most crucial in which the mappers decompose itemsets ,the reducers perform combination operation by constructing small ultrametric trees and actual mining of these trees take place separately. The mining of FIUT takes place in two phases. The first phase of Mining involves two rounds of scanning. Two MapReduce jobs takes place in this phase. The second phase involves construction of K-FIU trees and discovery of K-itemsets and is handled by third MapReduce. In this way by incorporating FIUT trees we improve the parallel mining of Frequent itemsets.

KEYWORDS: frequent itemset mining , MapReduce , parallel mining , ultrametric tree.

I. INTRODUCTION

As Association Rule Mining follows a particular procedure which is meant to find frequent patterns , correlation , association from datasets such as relation , transaction databases. Example: In real world, when a customer purchases a sandwich it is likely to buy a ketchup along. This is exactly how the association rule mining works.

In case of sequential pattern mining it is a process of connecting a topic of data mining with identifying the similar patterns . When these are put in use a problem occurs in Frequent Item set Mining(FIM)-Is a method or a process which takes place in a particular way for example: an artist prefers to paint the background first and then filling in the details , therefore this pattern is followed frequently by him. FIM creates fragments of mining time of a particular portion , this is done due to high input or output intensity. Because of which it is necessary to speed up the process , which is difficult to achieve .By introducing FIM which uses MapReduce to solve the issue i.e., when a dataset in data mining application is huge the sequential FIM algorithm running on a single machine results is catastrophic. Therefore MapReduce is used , it is used for processing large datasets by paralleling them amongst the computing nodes of a cluster. By optimizing the parallel FIM , it results in load balancing . Apriori and FP-growth are the categories of FIM. The apriori generates list of candidates list , using the bottom up approach it scans for the frequent item sets and groups the frequently used candidates list. To reduce the time taken for scanning FP-growth algorithm was introduced which is scalable and efficient , it compresses the storage by constructing the prefix tree , which eliminates the generation of candidates and saves the time which is required for scanning. The disadvantage of FP-growth is that it is infeasible in constructing the in memory FP tree, this becomes even difficult when it comes to multi dimensional database. To overcome these faults the frequent items ultrametric tree (FIU-tree) is used due to its advantages like reducing the input or output overhead, offering a natural way of partitioning a dataset, compressed storage , recursively traverse and also enables automatic parallelization, load balancing, data distribution, and fault tolerance on large computing clusters which was lacking in previously used algorithms. To solve the above mentioned problems we incorporate a parallel

ISSN(Online) : 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

An ISO 3297: 2007 Certified Organization Vol.5, Special Issue 5, June 2017

8th One Day National Conference on Innovation and Research in Information Technology (IRIT- 2017)

Organized by

Departments of ISE, CSE & MCA, New Horizon College of Engineering, Bengaluru, Karnataka 560103, India

mining FIM algorithm called FIDOOP using MapReduce. In FIDOOP we construct small ultra metric trees which is one of the main advantage.
The summary of the paper's contribution are as follow:

1. Overcoming the problems seen in FIUT parallelizing.
2. Developing FIDOOP using MapReduce.
3. Introducing data distribution method for load balancing.

II.RELATED WORK

The Apriori algorithmic program could be a classic approach of mining frequent itemsets in a database[1]. A spread of Apriori-like algorithms aim to shorten database scanning time by reducing candidate itemsets. for instance, Park et al projected the direct hashing and pruning algorithmic program to manage the amount of candidate two-itemsets and prune the information size employing a hash technique, within the inverted hashing and pruning algorithmic program [2], each k-itemset at intervals every dealings is hashed into a hash table.

To improve the performance of Apriori like algorithms ,Han etal[3] planned a completely unique approach known as FP-growth to avoid generating associate excessive range of candidate itemsets.

The problems associated with FP-growth are:

1) the construction of a giant range of conditional FP trees residing within the main memory and
2) the algorithmic traverse of Fp-trees to handle this downside.

Tsay et al. [4]planned a brand new technique known as FIUT, that depends on frequent items ultrametric trees to avoid recursively traversing FP trees. Zhang et al. [5] planned a method of strained frequent pattern trees to considerably improve the potency of mining association rules.

III. PROPOSED ALGORITHM

I. Preliminary details

A. *Association rules*

Association rule mining helps in decision making by identifying patterns in a database and forming rules which support the decision. Association rule mining is widely used in the medical field and supermarkets.The application in supermarkets is commonly known as market basket analysis. An example of ARM is prediction of heart disease of a person based on his blood pressure and his exercise patterns. The rule which is formed consists of two parts antecedent and the consequent .For Eg if a person buys bun and vegetables , a rule could be formed that he will buy patty for making a burger buys(bun, vegetables)=>buys(patty).The association rule mining performance measures are Confidence, support , Minimum support threshold and minimum confidence threshold..A subset of frequent itemset should also be frequent ie.If
{AB} is a frequent itemset ,both {A} and {B} should be a frequent itemset. The main objective of ARM is to identify rules which satisfy the minimum support and count.

B. Frequent itemset ultrametric tree

The FIUT approach is mainly used to identify Frequent item sets in large databases. FIUT uses ultametric trees to improve the efficiency of frequent itemset identification. The FIUT is structured as follows.

Firstly any tree construction starts with the root node, in our project the root node is the category of purchase by the customer. For example if a customer buys a product online under the electronics section then the root of this tree will be Electronics. Then the children of the tree can be Q_1,Q_2,Q_3 and so on. The items are the products which are brought under that category electronics, it could be a laptop, TV etc. The frequent items are inserted as the path from the root and the nodes are not repeated. We begin traverse from child Q1 of root and end with the leaf Q_m of the tree.

ISSN(Online) : 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

An ISO 3297: 2007 Certified Organization Vol.5, Special Issue 5, June 2017

8th One Day National Conference on Innovation and Research in Information Technology (IRIT- 2017)

Organized by

Departments of ISE, CSE & MCA, New Horizon College of Engineering, Bengaluru, Karnataka 560103, India

Secondly it should be constructed in a way such that all the leaf nodes of the tree are in the same height. The speciality of FIUT tree is that the leaf nodes consists of two fields the name of the frequent item set and the counter indicating the number of transactions which included the item set. The non leaf nodes again consists of two fields that is the item name and a node link which connects is a link connecting a node to its child node.

This algorithm consists of two main phases ,the first phase involves the scanning of the database twice. The first generates a frequent one itemset by just scanning the entire itemset once. For e.g. if a database consists transaction details of a TV , laptop and refrigerator. All the items are counted once .The second scan ignores all the less frequently occurring items and prunes it. In this scan k-item sets are generated where k is the indication of the number of frequent items in the transaction. Phase two also involves two main operations, firstly k-FIU-trees are generated and then these k-item sets are mined based on its leaf without actually traversing the tree.

In most cases there is a comparison between the Fp growth and the ultrametric trees. Ultrametric trees outperforms the Fp growth by minimizing the input output overhead as it restricts the scanning to just two rounds. It also reduces the search space by partitioning the database efficiently. Importance is only given to the frequent item sets as all the infrequent items are removed and are not inserted into the tree which results in compressed storage. The computing time is also reduced by not traversing the tree every time

C . MapReduce Framework

MapReduce is one of the most promising and widely used programming model for applications involving large quantities of data and scientific analysis. MapReduce helps in computation as parallel operations on the key/Value pairs. MapReduce has two phases firstly the Map then reduce. The Map phase takes large amount of data as input and splits them into fragments ,the input is in the key/Value format. Once the fragments are formed these are distributed across the nodes of the cluster to process. After this the MapReduce runtime system groups and sorts the intermediate values formed after the map phase and later this is provide to the reduce tasks. Map Reduce is widely accepted and fault tolerant framework widely used by Google, Yahoo etc.

II.*Description of the Proposed Algorithm:*

We use the MapReduce programming model to implement a frequent itemset mining algorithm called FIDOOP. Using the algorithm we strive to achieve the goals such as automatic parallelization, load balancing and data distribution which were the main problem faced by traditionally used algorithms.

Storage is very expensive and important, to use minimum storage and avoid building conditional pattern bases we use FIUT trees. The challenge faced in this approach is converting the serial FIUT algorithm into parallel. The FIUT algorithm first involves generation of h-itemsets using the Data and Minimum support. Once the h-itemsets are generated, an iterative process is repeatedly performed until the loop variable k runs from M to 2.The construction of K-FIU trees and discovery of frequent k-itemsets are executed in sequential way. The worst part is, it is significant to construct K-FIU trees which is time consuming. This is shown in Algorithm 1(a).

ISSN(Online) : 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

An ISO 3297: 2007 Certified Organization Vol.5, Special Issue 5, June 2017

8th One Day National Conference on Innovation and Research in Information Technology (IRIT- 2017)

Organized by

Departments of ISE, CSE & MCA, New Horizon College of Engineering, Bengaluru, Karnataka 560103, India

Algorithm 1 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-FIU-tree);
 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 ($k + 1 \le h \le M$) 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 1(B) actually constructs the K-FIU tree by decomposing the h itemset into all possible k itemsets, where k+1 < h <= M and finally we are going to union original k-itemsets. The decomposition of h itemsets is performed in a sequential way ,usually from long to short itemsets.

The serial FIUT algorithm is improved in the following two phases:

1. The first phase of FIUT scans the database twice as two MapReduce jobs. The first round of scanning produces frequent one itemsets. The second round of scanning generates k-itemsets by pruning the infrequent items.
2. The second phase involves construction of k-FIU trees and discovery of frequent k-itemsets which is done by third MapReduce job. The h itemsets where (2< h<=M) are decomposed into (h-1) itemsets,(h-2) itemsets...two itemsets. In third MapReduce job the generation of short and long itemsets is independent of each other. These two steps solves the parallelization problem faced by Algorithm 1(A) & 1(B).

Algorithm 2 ParallelCounting: To Generate All Frequent One-Itemsets

```
Input: minsupport, DBi;
Output: 1-itemsets;
 1: function MAP(key offset, values DBi)
 2:     //T is the transaction in DBi
 3:     for all T do
 4:         items ← split each T;
 5:         for all item in items do
 6:             output( item, 1);
 7:         end for
 8:     end for
 9: end function
10: reduce input: (item,1 )
11: function REDUCE(key item, values 1)
12:     sum=0;
13:     for all item do
14:         sum += 1;
15:     end for
16:     output(1-itemset, sum); //item is stored as 1-itemset
17:     if sum ≥ minsupport then
18:         F − list ← the (1-itemset, sum) //F-list is a CacheFile storing frequent 1-itemsets and their count.
19:     end if
20: end function
```

ISSN(Online) : 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

An ISO 3297: 2007 Certified Organization *Vol.5, Special Issue 5, June 2017*

8th One Day National Conference on Innovation and Research in Information Technology (IRIT- 2017)

Organized by

Departments of ISE, CSE & MCA, New Horizon College of Engineering, Bengaluru, Karnataka 560103, India

Algorithm 2 explains the first MapReduce which discovers all frequent one items. The input to this algorithm is the entire database and the output is Frequent one itemsets. The second MapReduce to generate k itemsets by pruning is shown in Algorithm 3.

Algorithm 3 Generatekitemsets: To Generate All *k*-Itemsets by Pruning the Original Database

```
Input: minsupport, DBi;
Output: k-itemsets;
 1: function MAP(key offset, values DBi)
 2:     //T is the transaction in DBi
 3:     for all (T) do
 4:         items ← split each T;
 5:         for all (item in items) do
 6:             if (item is not frequent) then
 7:                 prune the item in the T;
 8:             end if
 9:             k-itemset ←(k, itemset) /*itemset is the set of frequent items
    after pruning, whose length is k */
10:             output(k-itemset,1);
11:         end for
12:     end for
13: end function

14: function REDUCE(key k-itemset, values 1)
15:     sum=0;
16:     for all (k-itemset) do
17:         sum += 1;
18:     end for
19:     output(k, k-itemset+sum);//sum is support of this itemset
20: end function
```

The last MapReduce is the most difficult one and is mainly topic of discussion in this paper. The third MapReduce constructs K-FIU tree and is used to mine all frequent k-itemsets. Input to this stage is minimum support and the database. This stage is responsible for decomposing, constructing and mining of the FIUT tree. For eg if each itemset has k items and initially the value of k is M. The Mappers decompose h-itemsets where 2< h<=M into (h-1)(h-2)... and two itemsets. Multiple mappers are used for this purpose which makes the decomposition parallel and improves the storage as well as the efficiency. FIDOOP takes advantage of Hadoop runtime system , during the shuffling item numbers are output keys of key-value pairs produced by mappers. Finally there is no need of recursive mining; tree is discarded after mining.

Algorithm 4 Miningkitemsets: To Mine All Frequent Itemsets

```
Input: Pair(k, k-itemset+support);//This is the output of the second
    MapReduce.
Output: frequent k-itemsets;
 1: function MAP(key k, values k-itemset+support)
 2:     De-itemset ← values.k-itemset;
 3:     decompose(De-itemset,2,mapresult); /* To decompose each De-
    itemset into t-itemsets (t is from 2 to De-itemset.length), and store the
    results to mapresult. */
 4:     for all (mapresult with different item length) do
 5:         //t-itemset is the results decomposed by k-itemset(i.e. t ≤ k);
 6:         for all ( t-itemset ) 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 function

12: function REDUCE(key t, values t-FIU-tree)
13:     for all (t-FIU-tree) do
14:         t − FIU − tree ← combining all t-FIU-tree from each mapper;
15:         for all (each leaf with item name v in t-FIU-tree) do
16:             if ( count(v)/| DB |≥ minsupport ) then
17:                 frequent h − itemset ← pathitem(v);
18:             end if
19:         end for
20:     end for
21:     output( h, frequent h-itemset);
22: end function
```

ISSN(Online) : 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

An ISO 3297: 2007 Certified Organization Vol.5, Special Issue 5, June 2017

8th One Day National Conference on Innovation and Research in Information Technology (IRIT- 2017)

Organized by

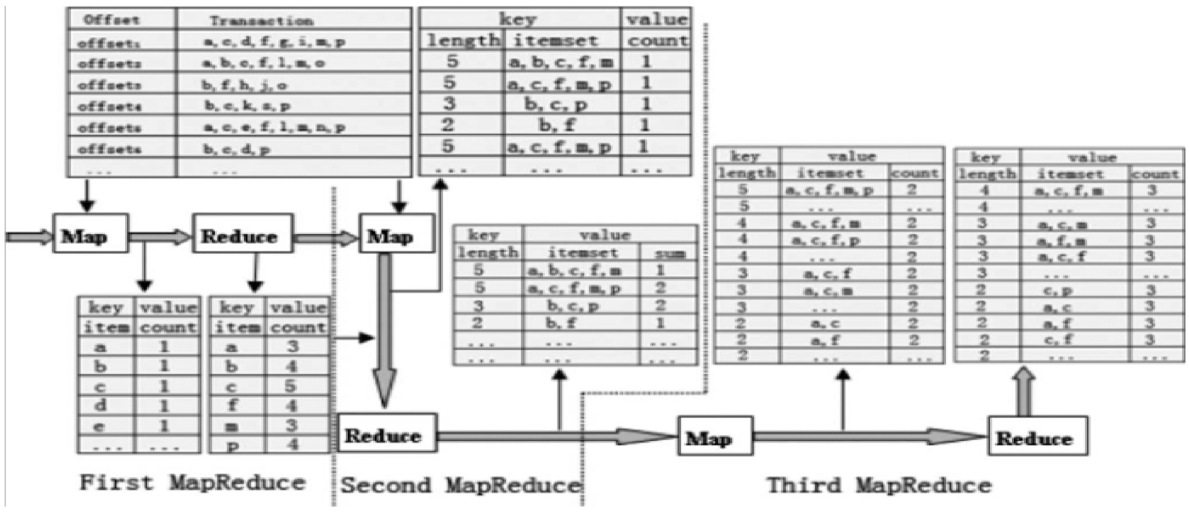
Departments of ISE, CSE & MCA, New Horizon College of Engineering, Bengaluru, Karnataka 560103, India

The key-value pairs generated after the third MapReduce job is used to construct the FIU tree as shown in Algorithm 4.In the key value pair key represents number of items in an itemset and value is FIU tree consisting of leaf and non leaf nodes. Using the key all itemsets having same number of items are delivered to single reducer. The decompose function is explained in algorithm 5 and is used for decomposing h-itemset into list of k itemsets.

Algorithm 5 Decompose(): How to Decompose a *k*-Itemset

```
Input: to be decomposed string s;
Output: the decomposed results l;
1: function DECOMPOSE(s, l, de-result)
2:     /* s is the string to be decomposed, l is the length of the itemset
   required to be decomposed, de-result stores the results.
3:     for all ( i is from l to s.length) do
4:         decompose(s, i, result, resultend);
5:         de-result ← i+resultend;
6:     end for
7: end function
8: function DECOMPOSE(s, m, result, resultend)
9:     if (m == 0) then
10:        resultend.addAll(result);//resultend is a list storing all the i-item
   set
11:        return;
12:    end if
13:    if (s is not null) then
14:        result.add(s[0]+null);
15:        for all (j is from the second value to the last value of s) do
16:            $s1[j-1] \leftarrow s[j]$;
17:        end for
18:        decompose(s1, m - 1, result, resultend); //when selecting the first
   item
19:        result.remove(result.size() - 1);
20:        desompose(s1, m, result, resultend); //when the first item is not
   selected
21:    end if
22: end function
```

Overview of MapReduce-based FIDOOP



Offset	Transaction
offset1	a, c, d, f, g, i, m, p
offset2	a, b, c, f, l, m, o
offset3	b, f, h, j, o
offset4	b, c, k, s, p
offset5	a, c, e, f, l, m, n, p
offset6	b, c, d, p
...	...

key	value
item	count
a	1
b	1
c	1
d	1
e	1
...	...

key	value
item	count
a	3
b	4
c	5
f	4
m	3
p	4

key		value
length	itemset	count
5	a, b, c, f, m	1
5	a, c, f, m, p	1
3	b, c, p	1
2	b, f	1
5	a, c, f, m, p	1
...	...	...

key	value	
length	itemset	sum
5	a, b, c, f, m	1
5	a, c, f, m, p	2
3	b, c, p	2
2	b, f	1
...	...	...
...	...	...

key	value	
length	itemset	count
5	a, c, f, m, p	2
5	...	...
4	a, c, f, m	2
4	a, c, f, p	2
4	...	2
3	a, c, f	2
3	a, c, m	2
3	...	2
2	a, c	2
2	a, f	2
2	...	...

key	value	
length	itemset	count
4	a, c, f, m	3
4	...	...
3	a, c, m	3
3	a, f, m	3
3	a, c, f	3
3	...	...
2	c, p	3
2	a, c	3
2	a, f	3
2	c, f	3
2	...	...

IV. PSEUDO CODE

Step 1: Identify all frequent one itemsets using first MapReduce job.
Step 2: prune all infrequent items from the transactions.
Step 3: Perform third MapReduce job to decompose itemsets and construct the K-FIU trees
Step 4: The last stage of MapReduce is performed which involves mining of ultrametric trees seperately.

ISSN(Online) : 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

An ISO 3297: 2007 Certified Organization Vol.5, Special Issue 5, June 2017

8th One Day National Conference on Innovation and Research in Information Technology (IRIT- 2017)

Organized by

Departments of ISE, CSE & MCA, New Horizon College of Engineering, Bengaluru, Karnataka 560103, India

Step 5:Once mining is completed discard the k-FIU tree.
Step 6:Continue the process .Go to step 1
Step 7: End.

V. IMPLEMENTATION

Load balancing is implemented as follows:

(A)Load Balance

Decompose function of Third MapReduce depends on the length of itemset. The decomposition cost exponentially increases with increase in length of itemset.A workload balance metric is used to balance load among nodes. If database p is partitioned across p data nodes then $p_i(\mathrm{IS}_m) = (C_i(\mathrm{IS}_m)/\sum_{j=1}^{p} C_j(\mathrm{IS}_m))$ probability that node i contains itemset and length is m. IS_m denotes set of itemsets where length of each itemset is m ,$C_I(\mathrm{IS}_m)$ is count of IS_m is node i.The weight is calculated to specify the load of IS_m.

Computing -load weight of IS_m over all itemset is as :

$$\omega(\mathrm{IS}_m) = \frac{C(\mathrm{IS}_m) \times 2^m}{\sum_{j=2}^{M} C(\mathrm{IS}_j) \times 2^j} \qquad (1)$$

2^m is time complexity for decomposing m-itemsets. $C(\mathrm{IS}_m)$ is the count of IS_m over datanodes. In a transaction database D partitioned over p nodes and random itemset Y, the computing load is given by

$$W_i = \sum_{X \subset \mathrm{IS}} \omega(X) \times p_i(X). \qquad (2)$$

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

Data distribution leads to high load-balancing performance if the weights W_i ($i \in [1, p]$) are identical. where as when they are different W_i leads to poor load balancing. Entropy is used as the load balance metric. For database D the load balance metric is expressed as.

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

The WB(D) metric is defined in terms of entropy and has the following properties.
1.If WB(D) equals 1,decomposition load is perfectly balanced across all the nodes.
2.If WB(D) equals to 0,decomposition load is balances on one node.
3.All cases represented by $0 < \mathrm{WB(D)} < 1$.

VI. CONCLUSION AND FUTURE WORK

A parallel frequent itemsets mining algorithm called FiDoop is implemented using the MapReduce model which resolves the load balancing and the scalability issues seen in the existing parallel mining algorithm. The performance of FiDoop is improved by balancing the input or output loads on the clusters of data nodes. The traditional FP trees are

ISSN(Online) : 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

An ISO 3297: 2007 Certified Organization *Vol.5, Special Issue 5, June 2017*

8th One Day National Conference on Innovation and Research in Information Technology (IRIT- 2017)

Organized by

Departments of ISE, CSE & MCA, New Horizon College of Engineering, Bengaluru, Karnataka 560103, India

replaced by the ultra metric trees , which results in compressed storage and avoids in building the conditional pattern bases. Three MapReduce jobs are incorporated in the MapReduce model , the third phase plays an important role in the parallel mining operation. Mappers are responsible for the construction of small ultra metric tree for separate mining.
In further research will apply a metric to measure the load balance , this metric is applied to investigate advanced load balance strategies in the form of FiDoop. We will incorporate FiDoop with the data-placement mechanism on heterogeneous clusters. We also aim at investigating the impact of heterogeneous data placement strategy on Hadoop-based parallel mining of frequent itemsets , and also the performance issues, efficiency of energy and thermal management.

REFERENCES

[1] R. Agrawal, T. Imielinski,´ and A. Swami, “Mining association rules between sets of items in large databases,” *ACM SIGMOD Rec.*, vol. 22, no. 2, pp. 207–216, 1993.
[2] J. D. Holt and S. M. Chung, “Mining association rules using inverted hashing and pruning,” *Inf. Process. Lett.*, vol. 83, no. 4, pp. 211–220, 2002.
[3] J. Han, J. Pei, Y. Yin, and R. Mao, “Mining frequent patterns with-out candidate generation: A frequent-pattern tree approach,” *Data Min. Knowl. Disc.*, vol. 8, no. 1, pp. 53–87, 2004.
[4] Y.-J. Tsay, T.-J. Hsu, and J.-R. Yu, “FIUT: A new method for mining frequent itemsets,” *Inf. Sci.*, vol. 179, no. 11, pp. 1724–1737, 2009.
[5] J. Zhang, X. Zhao, S. Zhang, S. Yin, and X. Qin, “Interrelation anal-ysis of celestial spectra data using constrained frequent pattern trees,” *Knowl.-Based Syst.*, vol. 41, pp. 77–88, Mar. 2013.