International Journal of Innovative Research in Computer Science & Technology (IJIRCST)
ISSN: 2347-5552, Volume-5, Issue-3, May 2017
DOI: 10.21276/ijircst.2017.5.3.2

# Privacy-Preserving in FiDoop, Mining of Frequent Itemsets from Outsourced Transaction Databases

**Nithin C, A V Krishna Mohan**

***Abstract*— Distributed computing has helped enthusiasm for a worldview called Datamining-as-a-service. This framework is useful for the organizations lack in specialized persons and processing asset empower to compute ,it enforces them to outsource their information to the outsider for data mining undertakings. In this work, we examine the strategy to safeguard security for frequent itemset mining in outsourced exchange databases and utilizing FiDoop calculation to process the infrequent itemsets from the outsourced datasets on the server side. A novel strategy used to accomplish k-support anonymity in light of measurable perceptions on the datasets to safeguard the security of the outsourced dataset. To decide frequent itemset at the distributed computing side from the database that gotten from various associations by utilizing a parallel mining for frequent itemsets, algorithm called FiDoop utilizing the programming model of MapReduce. FiDoop coordinate frequent itemset ultrametric (FIU)tree rather than customary FP-tree to empower the packed stockpiling of the mined information and to stay away from conditional pattern based information. Three MapReduce tasks are utilized to mine the information from the outsourced data. In the significant third MapReduce undertaking, the mappers autonomously break down itemsets that delivered from second MapReduce, the reducers perform blend operations by building FIU-tree.**

***Index Terms*— Data mining, Frequent itemsets, Frequent itemset ultrametric tree (FIU-tree), Outsourcing and MapReduce.**

## I. INTRODUCTION

Data mining have become crucial in research and commercial environments, preserving the privacy of the individual data is very essential. The objective of successive itemset mining is to distinguish strong mining in the big data.

Information proprietor discovers reasonable to offload information for the outsider for computational reason, this procedure is known as outsourcing (Fig. 1).

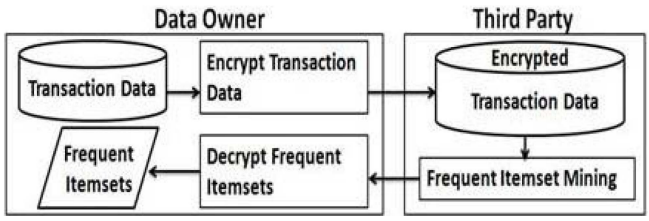


Fig. 1. Schematic plan for Mining of Frequent Itemset in outsourced data.

Here, the information proprietor or customer gives the information that to be mined; the protection ought to be kept up at the other computational site. Thus the customer encodes the information and after that offers it to the data miner. On the computing side, information mining on the encrypted data that transmitted for frequent itemsets utilizing FiDoop is done and the determined frequent itemset is sent back to the data owner. The original itemset is retained by decrypting the itemset obtained by the data miner.

To save privacy in mining of frequent itemset in outsourced data, the information proprietor must preprocess the information; hence the outsider data miner will be unable to gather/concentrate more data than it has given to the information proprietor. The past experience of the service provider with the information of the support of the computed frequent itemset, the third party should not enable to extract the other information which may lead to the privacy violation.

To maintain a strategic distance from such assaults, there exist many gathering based methodologies which fundamentally conceal the individuality of particular items with a group of itemsets. There are numerous strategies to acquire this approach; they are 1-diversity, k-anonymity, t-closeness, and so forth. And additionally with regular databases, these methodologies can likewise be utilized with other sort of information, for example, information from social network and graphical information. k-support anonymity is the most utilize technique to maintain a strategic distance from such assaults, a variation of k-anonymity, implies that for a

Manuscript Received May 12, 2017
**Nithin C**, Department of Computer Science & Engineering, Siddaganga Institute of Technology, Tumakuru, India.
**A V Krishna Mohan**, Assistant Proffesor, Department of Computer Science & Engineering, Siddaganga Institute of Technology, Tumakuru, India.

obtained thing with its support, have k-1 different things with a similar support.

Existing attempts to safeguard privacy in frequent itemset mining utilizes the approach of k-support anonymity with straightforward substitution encryption. Fake things or exchanges has been added into the original database to get k- support anonymity coming about to acquire spurious tuples from the data miner. A novel mixture technique is executed to accomplish k-support anonymity by including and furthermore expelling a few occurrences of things to the itemset based. The fake things which must be included are depends the factual perception. The exhaustive investigations on genuine and additionally synthetic datasets demonstrate that our strategies are powerful and give direct privacy.

The data transmitted to the outsider to decide the frequent itemsets. In association rule mining (ARM), the core issue is to decide the regular itemsets by information mining. The time expended to decide the incessant itemset is critical because of its high calculation and info yield power. This paper MapReduce used to play out the FIM, MapReduce is a broadly embraced programming model for handling big data by exploiting the parallelism among nodes in cluster.

Customarily frequent itemsets mining was performed by two calculations specifically, Apriori and FP-development. Apriori is a great calculation used to produce extensive number of hopeful itemsets by utilizing the affiliation controls by checking the support of the things in the itemset. The adaptability issue happened by the usage of FP-development like parallel FIM calculations. A noteworthy burden of FP-growthlike parallel calculations is infeasible to build FP trees for some vast scale databases.

This issue turns out to be exceptionally critical when it's up to a multidimensional and tremendous database. As opposed to run with Apriori and FP-development its advantageous to suggest frequent itemset ultrametric(FIU) tree for deciding the incessant itemsets on the distributed computing side. FIU-tree has four imperative components offering a characteristic method for partitioning a dataset, lessen input output overhead compacted stockpiling and keeping away from recursively transverse. The current parallel calculations do not have an instrument that empowers programmed parallelization, stack adjusting of data, data conveyance, and adaptation to non-critical failure on expansive computing clusters.

In FiDoop, the mappers freely and simultaneously break decompose itemsets; the reducers perform blend operations by building small ultrametric trees and also mining these trees in parallel.

## II. APPROCHED DESIGN

This segment depicts the proposed half and half strategy, that to include a few things and furthermore expel a few things into the genuine dataset. The existing method uses the k anonymity with all given k-1 itemset of same support. By implementing pseudo-taxonomy scheme, the real data base wont disturbs instead of that it adds fake dummy items. The items are clustered into groups of k by Frugal scheme clusters. The items in the group are sorted from the descending order of the support, the support of every transaction done equal to the first item of the dataset by adding duplicate items. Our technique is like the fugal scheme like we encode the original data of each transaction.

### *A. Encryption Algorithm*

In this algorithm, every transaction those transacted are modified hence the privacy is achieved. The encryption algorithm presented in shown (Algorithm 1). After generation of item table according to support and sorting sorting them in reducing order, we handle the exceptions in the IST by including fake things. For each exception, fake items are added with support equivalent to the support of the anomaly and they are added to the IST. To add them to the database, fake exchanges of various length, the fake exchange is divided into smaller exchanges of expanding sizes and add them to the encoded database as how many times it requires. For instance, if the length of fake exchange is of length 8, then it is isolated into littler exchange of size 2,6 and 2.

Table I. Notation

| Notation | Explanation |
|---|---|
| $T\,DB$ | *a transactional database* |
| $k$ | *k-support anonymity parameter* |
| $min\_sup$ | *the minimum support count threshold* |
| $E-T\,DB$ | *an encrypted transactional database* |
| $a_j, j = 1..k$ | *real items in a group of k* |
| $s_j, j = 1..k$ | *support of real items in a group of k items* |
| $d_{max}$ | *maximum absolute difference in Items removed* |
| $I_{Removed}$ | *set of all items removed* |
| $G_p, p = 1....n/k$ | *Group of k items* |
| $median_{Gp}$ | *median support of a group* $G_p$ |

For items having positive distinction, the frugal scheme is utilized by including fake transaction comprising of genuine items. For items having the negative contrast, examples of those things are expelled from transaction data base. This is done to lessen the impact of a item's removed from an transaction, to minimize the impact of removal while calculating the frequent itemsets. Then each item is replaced with its corresponding one-to-one mapping.

 

International Journal of Innovative Research in Computer Science & Technology (IJIRCST)
ISSN: 2347-5552, Volume-5, Issue-3, May 2017
DOI: 10.21276/ijircst.2017.5.3.2

**Algorithm 1**. Encryption Algorithm.

**Require**: TDB, k, min_sup.
**Ensure**: E - TDB, S, $I_{removed}$
1: Generate IST and sort it in descending order.
2: Detect outliers (if any).
3: **for** each of the outliers **do**
4: Add the required # of dummy fake items into 1STwith supports equal to the outliers support.
5: Construct a fake transaction consisting of the dummy items.
6: **end for**
7: **for** each fake transaction **do**
8: Divide the fake transaction of increasing sizes starting from 1, 2 ...
9: Add each of them as separate transaction to E-TDB, corresponding outlier support number of times.
10: **end for**
11: Cluster the items into groups of k.
12: **for** each group $G_p$ **do**
13: **for** each item $a_j$ in group $G_p$ **do**
14: difference($A_i$) = median$G_p - S_i$.
15: **if** | difference($A_i$) | > d max **then**
16: Find 1 <= l <= k such that $|S_l - S_L|$ <= $d_{max}$.
17: Calculate difference($A_i$) = $S_L - S_i$.
18: **end if**
19: **end for**
20: **end for**
21: Sort the differences in descending order.
22: **for all** items **do**
23: **if** $D_i$ > 0 then
24: Add fake transactions and generate synopsis.
25: **end if**
26: **if** $D_i$ < 0 then
27: Remove di instances of item ai from those transactions in TDB
which have item ai and has least cardinality.
28: **end if**
29: **end for**
30: Generate a one-to-one mapping function M(.).
31: Replace each item ai with M($A_i$).
32: return E - TDB and synopsis.
**end**

### *B. Decryption Algorithm*

The description algorithm is represented (Calculation 2). We initially decrypt the frequent itemset utilizing the coordinated mapping. In the event that the itemset contains a fake thing, we drop the itemset. On the off chance that it doesn't contain even one thing from the set 1relnoved, it brings about two cases. In the first case, if the support of itemset is not as much as min_sup, again we drop the itemset. Second, if the support is more noteworthy than min sup, we utilize the summation to discover whether it is a genuine itemset. On the off chance that the itemset contains no less than one thing from the set 1relnoved, then as opposed to doing a local mining on the transaction from which we expelled examples of things, for each stamped transaction, we increase the support of the itemset on the off chance that it is an appropriate subset of the stamped transaction. This is again trailed by a check whether the new support is more noteworthy than min-sup or not.

**Algorithm 2.** Decryption Algorithm

**Require:** FrequentItemsets ,synopsis, min_sup, $I_{removed}$
**Ensure:** TrueItemsets
1:**for** each itemset **do**
2: Decrypt the itemset using the mapping M(.).
3: **if** itemset contains a dummy item **then**
4: Drop the itemset.
5: Goto to line 1.
6: **end if**
7: **if** itemset contains at least one element from $I_{removed}$ **then**
8: **if** itemsetsupport < min_sup then
9: Drop the itemset.
10: Goto line 1.
11: **else**
12: Use synopsis to find whether it is a true itemset or not
13: **end if**
14: **else**
I5: For each marked transaction in TDB increment the support of the itemset by 1 if it is subset Of the marked transaction.
16: **if** support >= min sup then
17: Add the itemset to True Itemsets.
18: **else**
19: Drop the itemset
20: Goto line 1
21: **end if**
22: **end if**
23: **end for**
**End**

## III. OVERVIEW OF FIDOOP

We execute a parallel continuous itemsets mining calculation on the server side to called FiDoop to decide the frequent itemsets, from the data index that gotten from the client or information proprietor. FiDoop is a system fabricate work to acquire stack adjusting of data, autonomous parallelization and data dissemination of vast dataset for parallel mining of regular itemsets on large clusters. FiDoop utilizes the idea of FIU-tree in view of its element of enhanced data storage efficiency and to abstain from building conditional pattern of datasets, FiDoop utilizes FIU-tree to decide visit itemsets as opposed to customary FP trees. Parallelizing the serial algorithm FIUT is a test. In the Algorithm 3(A) after the creation of h-itemsets in first stage, an iterative procedure is run over and again to build k-FIU tree which used to decide the frequent itemsets of k length where the estimation of k differ from M to2. The k-FIU tree developing procedure and distinguishing of regular k-itemsets are done consecutively one after other. The k-FIU-tree is the FIU tree that built with the itemsets of length of k. k-FIU tree generation algorithm [Algorithm 3(B)] builds a k-FIU tree by breaking down every h-itemset, of length h into k-itemsets, where M ≥ h ≥ k+1. At that point, the union of unique k-itemsets with the k itemsets that decomposed by k+1 itemsets is figured to build the k-FIU tree. To produce k-itemsets, all conceivable h-itemsets (h > k) must be decomposed. Consequently, the process of decompose is consecutively finished with

 

itemsets of biggest length to most limited length. In that capacity, we enhance the serial FIUT algorithm as takes after

1) In initial segment of FIUT, checking of database is continue in the two Mapreduce frame, The principal MapReduce task is utilized to decide the frequent one-itemsets in the first round of filtering of the database. To get k-itemsets, the second Mapreduce needs to again check the itemsets from the database and need to prune the rare items from the transaction, which known by the continuous oneitemset from first MapReduce.

2) In the second phase of FIUT, incorporates the development of a k-FIU tree and third MapReduce task handles the recognition of frequent k-itemsets, where h-itemsets are exactly decomposed into a rundown of (h-1)-itemsets, (h-2)- itemsets, … , and two-itemsets. The creation of short itemsets in the third MapReduce task does not rely on upon the large itemsets. In option setting, short and long itemsets are delivered in parallel by parallel algorithm. Parallelization gotten by that approach in creation on itemsets for building tree. Algorithm 3(A) and (B) demonstrates the parallelizing techniques.

Proposed FiDoop incorporates three MapReduce tasks, they are explained in detail here. Every frequent items or frequent one-itemsets are distinguished by the principal MapReduce task (Algorithm 4). In this stage, database is the input for mapper and all one-itemsets is the input for reducer. The database is sought again to create k-itemsets in the second phase of MapReduce task, by eliminating the rare things in any transaction (see Algorithm 3). In the last MapReduce undertaking, which is the greatest troublesome of the three assignments incorporates the development of a k-FIU tree and handles the location of successive k-itemsets. Third MapReduce task is given the more consideration in this paper, as this is an performance barrier of the FiDoop algorithm.

**Algorithm 3** FIUT

```
1: function ALGORITHM 3(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 3(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
```

The three MapReduce tasks of our proposed FiDoop are portrayed in detail. The initial MapReduce work finds every regular itemset or successive one-itemsets ( Algorithm 4). In this stage, the input for Map errands is a database, and the yield of Reduce assignments is all regular one-itemsets. The second MapReduce task checks the database to produce k-itemsets by evacuating rare things in every transaction (Algorithm 3). The last MapReduce work—the most convoluted one of the three—develops k-FIU-tree and mines all regular k-itemsets. In this paper, we give careful consideration to the last MapReduce work, which is an execution bottleneck of the FiDoop.

**Algorithm 4** Parallel Counting

```
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
```

**Algorithm 5** Generating k-itemsets: pruning the infrequent items to generate k-itemsets.

```
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
```

Initial, an arrangement of mappers is taken care of by FiDoop, where each arrangement of mappers breaks down h-itemsets ($M \geq h \geq 2$) into a rundown of (h−1)- itemsets, (h

 

− 2)- itemsets, et cetera. what's more, two-itemsets. Performance in storing the data and input output ability is advanced by actualizing diverse mappers which breaks down h-itemsets in parallel. From that point forward, to join itemsets with the equivalent number having unique itemsets and to develop a k-FIU-tree from the itemsets deteriorated and used by mappers, FiDoop use diverse reducers. Full favorable position of the Hadoop runtime framework is taken by the FiDoop's tree-development technique, where the outcomes sets are replicated with the same keys delivered by mappers to a reducer by the rearranging stage. Amid the rearranging phase of FiDoop's third MapReduce undertaking, the thing numbers are been the yield keys of key-esteem sets used by the mappers. Finally, mining of k-FIU-tree is managed without having important to mine recursively. after continuous k-itemsets are mined, the k-FIU-tree is in a split second disposed of.

From the former clarification of the methods, it is demonstrated that via seeking the database, visit one-itemsets are in a split second made. Two-stage system is conveyed by FiDoop to develop a k-FIU-tree ($2 \leq k \leq M$) from k-itemsets. In the main stage, K-itemsets are secured by pruning rare things of any activities in the second hunt of database. The second stage is the combination of k-itemsets made by breaking down all h-itemsets (h>k). Keep in mind that the two-stage methodology is same as that of FIUT calculation, which guarantees the precision of new proposed calculation by this paper.

The progressive essential disclosure motivates to address an urgent issue relevant to balancing out load in FiDoop:

1. Large itemsets allow development for high-crumbling overhead
2. Short incorporated itemsets prompt numerous itemsets.

To accomplish extraordinary load changing viability, the constraints in the adjusting period of the MapReduce errands in FiDoop are joined together, there by settling the amount of itemsets across over reducers.

## IV. MAPREDUCE-BASED FIDOOP

The design issues of FiDoop built on the MapReduce framework has discussed in this section. Fig. 2 shows the working flow of FiDoop with three MapReduce phases. The results that produced after the third MapReduce are used to construct FIUtree (Algorithm 6).

### *A. First MapReduce Task*

The first MapReduce task is to generate all frequent one-itemsets that is frequent itemset of length one. The data of transactions that received are partitioned into multiple files, those files are stored across the data nodes in the Hadoop cluster, where it is managed by HDFS. Every transaction that received is stored in <LongWritable offset, Text record> format. Transaction feeds to every mapper from its input split. Then mappers compute the occurrences of each item in the itemset and generate local one-itemsets. After generating one-itemsets the reducer compare them with same key that comes from different mappers are merged and sorted in a specific reducer later which it produces a unique one-itemsets. Finally, infrequent items from one-itemsets are pruned that of with the count lesser than minimum support and consequently, global unique frequent one-itemsets are generated and in the format of<Text item, LongWritable count> as output from the first MapReduce job. The output from first MapReduce stored in a file name F-List which will be output to the second MapReduce. The pseudocode of first MapReduce job is shown in Algorithm 4.

Algorithm 4 represents pseudo code for the first MapReduce task.

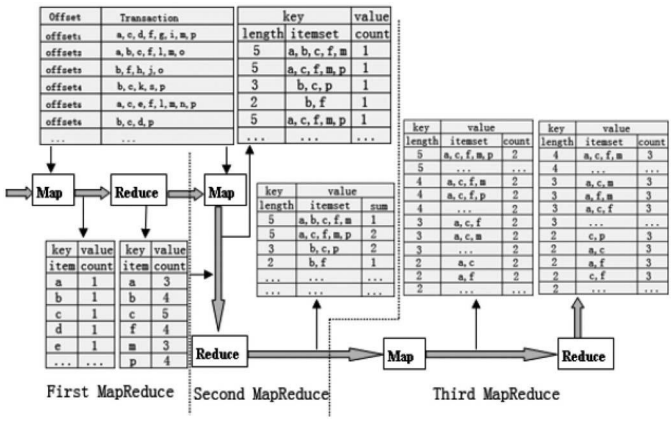


Fig. 2. Overview of MapReduce-based FiDoop

### *B. Second Mapreduce Task*

The second task of MapReduce, it does a moment round of scanning of the database to prune out the occasional items from every exchange record of data, those which decided in the first round of MapReduce. The second MapReduce task denote an itemset as a k-itemset ($M \geq k \geq 2$, where M is the maximum estimation of k in the pruned transaction). In the second Mapper, the itemsets those created are made out of the items exist in after pruning of the occasional itemsets from every set of data. In second Reducer, the reducer sort and combine the itemsets those created from the Mapper.

After the MapReduce operation, key/value sets generated from each reducer, where the key is the quantity of every itemset and the value is count of every itemset. Second MapReduce produces the yield with format <Int_Writable item_number, Map_Writable<Array_Writable k-items, Long_Writable Sum>>. Algrothim3 indicates the pseudocode for second MapReduce.

### *C. Third MapReduce Task*

The third MapReduce task is expensive in computationally stage. This stage is comprised of three jobs they are
1) decompose of items in itemsets,

 

2) building k-FIUT and
3) mining of frequent itemsets in database.

The work of every mapper is double in this stage, firstly it needs to decompose every k-itemset created by the second MapReduce task into a rundown of little estimated sets, the length of each itemset is inbetween 2 and k − 1 and furthermore to develop a FIU-tree by blending the results obtained with the itemset those produced from the second MapReduce with same length. The decomposition technique of every mapper is free of alternate mappers henceforth the scalability accomplished in the final Map-Reduce. Decomposition process done parallel by different mappers.

By constructing FIU-tree the data storage efficiency and I/O performance is improved, the merging of itemsets of same length in creation of tree helps to achieve the above described. The output from the third Mapper is a set of key/value pair, key represents quantity of items in that itemset and the value in FIU-tree denotes leaf & non-leaf nodes. Non leaf nodes denote item and node link, leaf-nodes denote item and its support. All the itemsets with the items of same length are sent to a one reducer. For example, to construct k-FIU tree, where value of k as 2, reducer takes the key value pair (k2, v2) to construct 2-FIU tree and frequent itemset mined by valuating the index of count from every leaf in k2-FIU tree, hence it does not require to repeatedly traverse through the tree to determine the frequent itemsets. Algorithm 6 represent the MapReduce functions, demonstrates to construct t-FIU-tree from t-itemsets. In Algorithm 6, the decompose() function is a repeatedly occurrence one, decomposing an h-itemset into a list of k-itemsets, k is length of itemset that may in-between 2 and h. Algorithm 7 has the pseudocode of decompose().

**Algorithm 6** 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 Deitemset 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**

**Algorithm 7** 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 from1 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-itemset
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: *s*1[ *j* − 1] ← *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**

## V. CONCLUSION

This paper illustrates a hybrid approach to preserve the privacy of the data, which has been outsourced to determine the frequent item from the third party. This paper also demonstrates to determine the frequent itemset at the cloud computing side with optimized characteristics like parallel computing, data distribution method across the clusters and load balancing among them, with a method called FiDoop. To give security through k-support anonymity, this

 

ISSN: 2347-5552, Volume-5, Issue-3, May 2017
DOI: 10.21276/ijircst.2017.5.3.2

technique includes and expels a few things. In this technique, the rate of impaction in creation of fake itemsets is reduced. The outsourced data that transmitted to the third-party cluster for the computational purpose of frequent item set are determined by using a efficient method called FiDoop, by using MapReduce processing technique. The datamining done by processing data through three MapReduce processes, the first two MapReduce used to prune the infrequent item from the transaction. The itemset from second MapReduce used to build frequent itemset ultrametric tree (fiut) by decomposing the itemsets. The frequent itemset determined by checking the leaf node count with minimum support values. FIUT reduces the workload burden of multiple scanning the data base since it requires scanning the data base only two times. The frequent itemset those determined by the third party are sent back to the data owner. Decryption of data received done by data owner to know the original data value set.

## REFERENCES

[1] 1 F. Giannotti, L. V. S. Lakshmanan, A. Monreale, D. Pedreschi, and W.H. Wang," Privacy-preserving mining of association rules from outsourced transaction databases.," IEEE Systems Journal, pp. 385-395, Vol.7,No.3,2013.

[2] L. Sweeney, "k-anonymity: A model for protecting privacy," International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems, Vol. 10, No. 5, pp. 557-570,2002.

[3] C-H Tai, P. S. Yu, and M-S Chen, "k-support anonymity based on pseudo taxonomy for outsourcing of frequent itemset mining," ACM SIGKDD international conference on Knowledge Discovery ,July 2010.

[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. Dean and S. Ghemawat, “MapReduce: Simplified data processing on large clusters,” Commun. ACM, vol. 51, no. 1, pp. 107–113, Jan. 2008.

 