International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395 -0056
Volume: 04 Issue: 04 | Apr -2017 www.irjet.net p-ISSN: 2395-0072

An Efficient Mining of Frequent Itemset Purchase on Retail Outlet using Frequent Itemset Ultrametric Tree on Hadoop

Manasa N[1], Venkatesh[2], Hemanth Kumar N P[3]

[1]PG Student, Department of CSE, Alva's Institute Of Engineering and Technology, Moodbidri, DK, India
[2]Associate Professor, Department of CSE, Alva's Institute Of Engineering and Technology, Moodbidri, DK, India
[3]Assistant Professor, Department of CSE, Alva's Institute Of Engineering and Technology, Moodbidri, DK, India

---

Abstract - *Mining frequent itemset purchase in a retail outlet is a very strenuous job. As extracting the frequently occurring itemsets over a large heterogeneous database is a core problem. Existing parallel mining algorithm for frequent itemsets does not support automatic parallelization, load balancing, synchronization and fault tolerance over a large cluster. Hence as a solution to these problem, an improvised algorithm called frequent itemset ultrametric tree algorithm using MapReduce programming model on Hadoop is used. Here in this technique, we implement three MapReduce jobs to perform the mining task.*

Key Words: **Frequent Itemsets, Frequent Item ultrametric Tre(FIUT), Hadoop, MapReduce, Hadoop Distributed File System(HDFS).**

1. INTRODUCTION

Finding out the frequent Item-sets in large heterogeneous database is one of the core problem in data mining [1]. Existing data mining algorithm like Apriori [2] and FP-growth [3] algorithm fails to extract frequent item-sets when size of transactional database is too large to compute. Also these traditional mining algorithms were running only on single machine which results in performance deterioration

Apriori is a bottom-up, breadth-first search algorithm. It uses hash trees to store frequent itemsets and candidate frequent itemsets. This classic Apriori- like parallel FIM algorithm uses generate and test process that generates a large number of candidate itemsets. The major disadvantage of Apriori-like FIM algorithm is that the processor has to repeatedly scan the entire database. To reduce the time taken for scanning entire database multiple times, an approach called FP-growth algorithm was introduced. Though FP-growth algorithm addresses the scalability problem it fails to construct in-memory FP trees to accommodate large-scale database. Hence rather than considering Apriori and FP-growth algorithm we incorporate a new frequent itemset mining algorithm called Frequent itemset ultrametric tree(FIU-tree) [4]. This FIUT algorithm is mainly used because of its advantageous features like, reduced i/o overhead, natural way of partitioning a dataset, compressed storage and avoids recursive traverse. Most importantly it enables automatic parallelization, load balancing, data distribution and fault tolerance over a large cluster. This FIUT algorithm is designed over MapReduce programming model [5]. Hence FIUT on Hadoop has got many distinctive features.

The MapReduce framework on Hadoop [6] enables distributed processing of huge data on large clusters, provided with good scalability, robust and fault tolerance. FIUT running on this framework is described by two major functions map and reduce. The mapper independently and concurrently decompose itemsets and on otherhand reducer function aggregates all the values by constructing small ultrametric trees as well as mining these trees in parallel. Industries utilize these extracted frequent itemsets in decision making about the products. If a retail sector company wants to know about the customer nature, their buying habits and about the product which is on demand this FIUT mining technique on Hadoop helps them to do so in very efficient way, in turn it increases their profit indeed. [7].

2. BACKGROUND STUDY

Hadoop is an open source software framework used for distributed storage and to develop data processing applications which are executed on those distributed computing environment where huge data sets are distributed across nodes in a cluster. Their main characteristic is to partition the data and computes it over large cluster of nodes. Hadoop includes various components such as Hadoop Distributed File System (HDFS), MapReduce, Hbase, HCatalog, Pig, Hive, Oozie, ZooKeeper, Kafka, and Mahout [5]. HDFS has become a key tool for managing pools of huge data and supports big data analytics application.

2.1 Hadoop Distributed File System

The Hadoop Distributed File System (HDFS) is designed for storing very large files with streaming data access patterns running on clusters of commodity hardware. Hadoop Distributed File System stores data to provide high aggregate I/O bandwidth [8]. HDFS stores filesystem metadata and application data separately where metadata is stored on a dedicated server called Namenode and application data are stored on other servers called DataNodes.

2.2 MAPREDUCE FRAMEWORK

MapReduce programming model is an associated implementation for processing and generating massive information sets with parallel, distributed algorithm rule on a cluster. Also this MapReduce programming model is an generic processing model that is used to address the general application problems. MapReduce computation can be seen is two phases, Map phase and Reduce Phases. During the Map section, the large input file is split into number of splits for analysis by map tasks running in parallel across the Hadoop cluster. By default, the MapReduce framework gets the input file from the Hadoop Distributed File System. The Reduce Phase aggregates all the values obtained from the Map task and generates a single output value. FIG- 1 shows the logical data flow in MapReduce. MapReduce greatly improves programmability by offering automatic data management, highly scalable, load balancing and fault tolerant processing.

Table -1: Symbol and Annotations

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

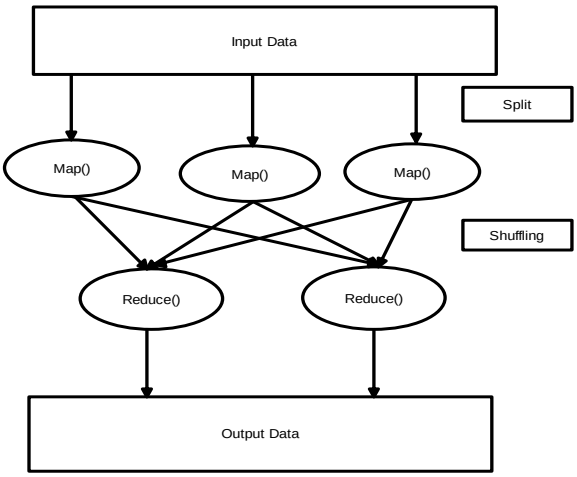

Fig -1: Data flow in MapReduce programming model

3. IMPLEMENTATION- FIUT ON HADOOP

The implementation of FIUT algorithm (see Algorithm 1) on Hadoop mainly targets to extract frequent item-sets. The Process constitutes of uploading the invoice input datasets obtained from the retail outlets in to the hadoop, preprocessing those inputs before uploading process and then applying the FIUT algorithm and generating the frequent itemsets.the architectural overview is depicted in the FIG 2. The FIUT algorithm consists of two mainly phases within two scans of database D. The phase one starts by computing the threshold value for all the items in the database. Later, Pruning technique is implemented to remove all infrequent items and remaining frequent items are used to generate k-itemset, where number of frequent items of a transaction is k in a database. Hence at the end of the phase one, all the frequent one-itemsets are generated. In phase two, small ultrametric tree are constructed repeatedly (see Algorithm 1(a) and 1(b).)

In this method to filter out the unrequired item-sets, we implement two methods namely Mapper and Reducer methods. Mapper sends all local frequent patterns whereas Reducer aggregates all the local frequent item-sets and calculates its number of occurrences. For those items which fail to reach the minimum threshold value, Reducer prunes them and keeps the remaining item-sets as the final outcome. This FIUT on Hadoop consists of three MapReduce jobs.

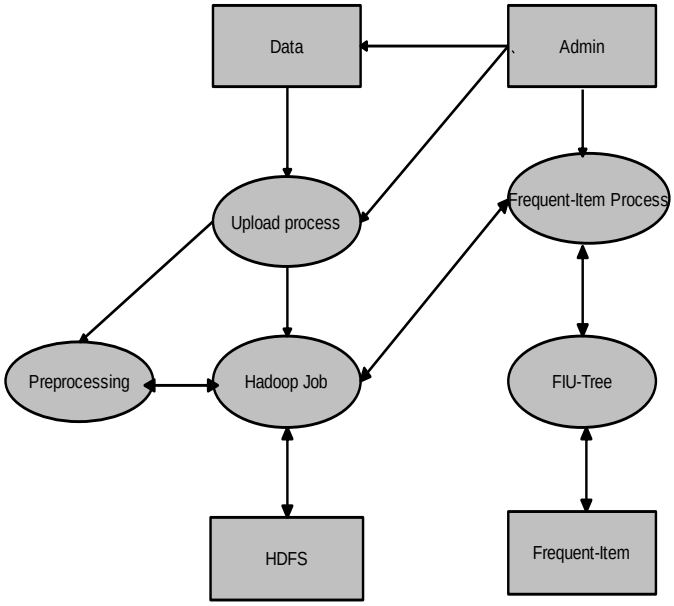

Fig -2: System Architecture of Proposed System

Algorithm 1 FIUT

ALGORITHM 1(A): FIUT(D, n)
Begin
1. h-itemsets = k-itemsets generation(D, Minthresh);
2. **for** k = Max down to 2 **do {**
3. k-FIU-tree = **k-FIU-tree generation (h-itemsets)**;
4. frequent k-itemsets Lk = frequent k-itemsets generation (k- FIUtree);
5. **}**
End

ALGORITHM 1(B): K-FIU-TREE GENERATION((h-itemsets))
Begin
1. Create the root of a k-FIU-tree, and label it as null (temporary 0th root)
2. **for all** ($k + 1 \leq h \leq Max$) **do**
3. decompose each h-itemset into all possible k-itemsets, and union original k-itemsets;
4. **for all** (k-itemset) **do**
5. ... build k-FIU-tree( ); *here, pseudo code is omitted*;
6. **end for**
7. **end for**
End

Algorithm 2 First MapReduce Job: To Generate All Frequent One- Itemsets

Input: Minthresh, 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**

1. **reduce input: (item,1)**
2. **function** REDUCE(key item, values 1)
3. sum=0;
4. **for all** item **do**
5. sum += 1;
6. **end for**
7. output(1-itemset, sum); //item is stored as 1-itemset
8. **if** sum ≥ Minthresh **then**
9. *F – list* ← the (1-itemset, sum) //F-list is a CacheFile storing frequent 1-itemsets and their count.
10. **end if**
11. **end function**

3.1 First MapReduce Job

The first MapReduce job identifies all frequent items or frequent one-itemsets. In this phase, the input of Map tasks is all frequent one-itemsets. Here, the transaction database is divided into many input files stored in HDFS across data nodes of a Hadoop Cluster. Here each mapper linearly reads each transaction from its local input split. The mapper calculates the number of occurrence of items and generates local one-itemsets. In turn these one-itemsets with the same key which is emitted by a different mapper are sorted and merged in a specific reducer. This reducer further produces the global one-itemsets. At last, the items which cannot meet the minimum threshold value are pruned off and accordingly, global frequent one itemsets are generated. The local file named F-list stores all frequent one-itemsets and their counts which in turn becomes the input to the second MapReduce job. Algorithm 2 illustrates the first MapReduce job in detail.

3.2 Second MapReduce Job

The second MapReduce job scans the datasets to generate k-itemsets by pruning the infrequent items in each transaction. This second MapReduce task applies a second round of scanning on the datasets to remove the infrequent items from each transaction data. The second job identifies the itemset as a k-itemset if it contains k frequent items. Here each mapper intake the transaction record and emits the itemsets with its count. These itemsets and its count are combined and shuffled which in turn fed as input to the reducer phase. After completion of the combination operation, each reducer will count. These itemsets and its count are combined and shuffled which in turn fed as input to the reducer phase. After completion of the combination operation, each reducer will emit the itemset and its count in terms of key/value pair. The pseudocode of second MapReduce job is Algorithm 3.

3.3 Third MapReduce Job

The MapReduce job is the most computational one which constructs k-FIU tree and mines all frequent k-itemsets. The main task of this third MapReduce job is to decompose the itemset, construct the k-FIU trees and to mine the frequent itemset [4]. The task of each mapper is to decompose each k-itemset in to a list of small-sized sets, which is obtained by the second MapReduce job and to construct an FIU tree by merging all decomposed results of same length. In this third MapReduce job, each mapper is independent of itself; many mapper can perform the decomposition process in parallel. The Map function of this job produce a number of items in an itemset along with an FIU tree that consists of leaf nodes and nonleaf nodes where nonleaf nodes consists of item name and node link, leaf nodes consists of item name and its support. The Reducer constructs k2-FIU-tree and mines all frequent itemsets only by checking the count value of leaf node in the k-2-FIU tree without traversing the tree recursively. An algorithm 4 depicts the Map and Reduce functions of third MapReduce job.

Algorithm 3 Second MapReduce Job: To Generate All k-Items by Pruning the Original Database

Input: Minthresh, 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

1.  function REDUCE(key k-itemset, values 1)
2.     sum=0;
3.     for all (k-itemset) do
4.          sum += 1;
5.     end for
6.    output(k, k-itemset+sum);//sum is support of this itemset
7.  end function
```

Algorithm 4 Third MapReduce Job: 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

1.  function REDUCE(key t, values t-FIU-tree)
2.     for all (t-FIU-tree) do
3.        t − FIU − tree ← combining all t-FIU-tree from each mapper;
4.        for all (each leaf with item name v in t-FIU-tree) do
5.           if ( count(v)/| DB |≥ Minthresh ) then
6.             frequent h − itemset ← pathitem(v);
7.           end if
8.        end for
9.     end for
10.  output( h, frequent h-itemset);
11. end function
```

4. CONCLUSIONS

To solve the problems of existing parallel mining algorithm for accessing the frequent itemset purchase in retail outlets, We applied a new technique of Mapreduce programming model to develop a parallel frequent itemsets mining algorithm called Frequent Itemset ultrametric tree implemented on Hadoop cluster. This method incorporates three MapReduce jobs to complete the parallel mining of frequent itemsets. The data is collected from UCI datasets which are preprocessed and uploaded into Hadoop .When frequent itemsets mining algorithm is invoked, the datasets are fetched from Hadoop and mining algorithms are processed on data and produced frequent itemsets. FIUT on hadoop has been dedicated to produce an accurate data mining results under Hadoop cluster environment.

REFERENCES

[1] C.Borgelt, "Frequent item set mining", *wiley intersisciplinary Reviews: Data Mining and Knowledg* Discovery, vol. 2, pp. 437-456, 2012.

[2] R. Agrawal, T. Imieli´nski, and A. Swami, "Mining association rules between sets of items in large databases," *ACM SIGMOD Rec.*, vol. 22, no. 2, pp. 207–216, 1993.

[3] J. Han, J. Pei, Y. Yin, and R. Mao, "Mining frequent patterns with- out candidate generation: A frequent 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. Xie, et al., " Improving mapreduce performance through data placement in heterogeneous hadoop cluster", in parallel and distributed processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE Internatioinal Symposium on, 2010, pp. 1-9.

[6] T. White, Hadoop: The definitive guide: "O`Reilly Media, Inc.", 2012.

[7] P. Zikopoulos and C. Eaton, Understanding big data: Analytics for enterprise class hadoop and streaming data: McGraw-Hill Osborne Media, 2011.

[8] M Bhandarkar, "MapReduce programming with apache Hadoop, "in Parallel & Distributed Processing(IPDPS), 2010 IEEE International Symposium on, 2010, pp. 1-1.