Integratingk-Nearest Neighbour Algorithm and Map Reduce on FiDoop

Vernon Louis, Jyothi S.

***Abstract*--Data has become one of the cornerstones of any organization that is involved in any field. The results interpreted from analyzing it help in an organization upward economic growth. Although useful, the raw data is often hard to understand and hence needs to be mined. This is further aggravated in the Big Data environment. The FiDoop is a development on the Hadoop concept, where only data sets that are accessed more often are considered to be of at most importance. Older methods, although successful in achieving parallelization, end up having a huge overload or long traversal tress through the database to provide results. FIU-tree is leveraged to handle efficient traversal through databases. In order to enhance the results a collection is first created using the k-NN algorithm. This is a 2-phase algorithm that uses a predefined value for 'k' and based on this it proceeds with further computations. When it comes to Big data, using k-NN alone would end up in generalization of datasets because of the way it computes the nearest neighbour. Hence the Map Reduce concept is implemented along with these algorithms.**

Keywords*-- *FiDoop, MapReduce, K-Nearest Neighbour, Frequent itemsets ultrametric tree, Apriori algorithm.

1) INTRODUCTION

Data mining is one of the most important factors in the present world given that day by day the size and collection of data is increasing indefinitely. This has led to the storage of a large amount of data of varied structures and types, that belong to various fields. Here in lies the importance of data mining. Frequent Itemsets mining (FIM)[1] has found its importance in the present environment of big data because of very obvious reasons. Traditional mining applications that rely on sequential FIM cannot handle the load as they run on single machines and eventually have degraded performance. This is attributed to the use of sequential algorithms which are linear and limited to working on a single entity at a particular instance. This is not suitable for the big data environment as the time required to go through the data sets linearly with the increase of the data base and this is not efficient. Therefore the most obvious solution which comes with the progress in parallel programming environment is parallel mining techniques. Now this would essentially in the form where an algorithm is run of 'n' number of machines simultaneously to fetch the results.

The MapReduce[1][7] programming concept is leveraged here to help achieve parallelization. Apache Hadoop[8] is one of the most widely used application that operates on the distributed computing environment. It has high computational efficiency and allows for increased flexibility. Hadoop makes major use of the MapReduce. Frequent Itemsets help reduce the overload of going through all the datasets in a system. It simply can be summarized like this, why go through the effort of scanning all the datasets when the best result can be found by scanning only those datasets that have the most accurate results. Apriori algorithms[1] and FP-growth[3] tree algorithms traversed the entire system to give back results but with additional overheads. Thus the FIU-tree takes over and solves the issue of load balancing as well as long traversal paths. In the following sections various data mining techniques will be looked upon and then a detailed overview of both the k-NN algorithm and the FIU-tree algorithm implementing the MapReduce functionality will be looked into.

2) MINING FREQUENT ITEMSETS

Mining is the process of identifying itemsets that have a particular pattern or a relationship between those itemsets. This detection of itemsets helps reduce the overhead of looking through all the entities that may or may not be related to the problem at hand. Though this is not relatively new, the tremendous increase in size has led to newer problems in this domain. The most widely used algorithms for data mining are Artificial Neural Networks, Decision trees and the nearest neighbour. This approach technically combines the decision tree method and the nearest neighbour algorithms to give out better efficiency. Since the problem it focuses is not datasets in general, but frequent itemsets, it is better to understand what this actually is.

Frequent itemsets can be generalized as those entities that have the most relevant data or those data sets that have occurred repeatedly in a number of instances. This will essentially be of priority as the chances of it being looked up are higher when compared to those that have hardly been in the picture.

Frequent itemset mining has its own set of algorithms to deal with. FIM algorithms can be divided into Apriori and FP-growth algorithms. Apriori is one of the older algorithms that uses the generate and test process that always resulted in a large number of candidate itemsets. One of the drawbacks of this is the time constraint involved as it needed to scan the entire database. A better approach to battle this was the FP-

growth which removed the candidate itemset creation process. This approach aimed at maintaining FP- trees. Even though this addressed the scalability issue of the Apriori algorithm, it resulted in maintaining in memory FP trees which is deemed infeasible because of the huge overhead it has on the memory which is further aggravated by the size of the dataset.

The drawbacks of the above mentioned algorithms needed a solution and hence the Frequent Item ultrametric tree (FIU-tree). This has 4 features that have made it prominent. Reduced I/O overhead has a natural way to dissect the dataset, compressed storage and averting recursively traversing the datasets.

3) PRELIMINARY

An overview of the k-NN[7] algorithm, FIU-tree algorithm are provided below. This

A. k-Nearest Neighbour algorithm

The kNN can be termed as a lazy learning algorithm. It is non-parametric in nature and does not work on any assumptions on the below database structure. This is a plus point of this algorithm because data does not obey any theoretical assumptions. kNN algorithm works on some prerequisites;

- it assumes that data is in a feature space or metric space.
- data can either be scalars or vectors and hence may have a sense of distance, not in the literal sense.
- it makes use of training data as a reference to distribute the new data points in a organized manner which is called the testing dataset

The kNN algorithm has 2 phases.

- the training phase
- the testing phase

1) The Training Phase

This phase generally involves storing the data with vector coordinates. This helps classify the datasets in the testing phase.

2) The Testing Phase

Now that we have data points , these are tested w=to find class label for a new point. Class label is an identifier for the data vector coordinates.

a) k=k or k nearest neighbour

In this algorithm , for a given data point from the testing data set, 'k' nearest neighbours are found and classified by the majority vote that is carried out in the training phase.

b)Generalising the k-Nearest Neighbour Testing Phase

- get value of 'k'
- prepare the training dataset
- retrieve data points from the testing dataset
- carry out majority vote on closest neighbours based on distance metric
- assign class label to majority winner
- repeat until all the data in the testing phase is classified.

B. FIUT Algorithm

The FIUT[1] makes use of the ultrametric tree to enhance the mining process. This tree is constructed in the following manner;

- The root is labelled as null. An itemset of frequent items is inserted with connecting edges beginning with the first child to the last leaf child with no duplicate nodes.
- All the itemsets are inserted as paths to for the FIU-tree and hence all the leaf nodes will be at the same height.
- The leaf nodes have two fields; item-name and count. Count of an item name represents the number of transactions that contain the itemset. Non leaf node contain wo fields as well, item-name and node-link. The node-link behaves as a pointer linking to the other child nodes present in the tree.

This algorithm also has two phase. The first phase scans the database twice. The first scan results in frequent one-itemsets by computing all the items present in that set. and the second scan prunes or removes all non frequent items in each transaction.

C. MapReduce Framework

MapReduce is a programming concept and an eventual implementation which helps process and generate large datasets. It works best on the Hadoop distribute file system(HDFS)[7]. It generally processes large volume of data which is residing on many machines. MapReduce breaks down a large dataset into individual independent chunks. The size of each resultant subset of data depends on the number of nodes available to process it along with the size of the data. This involves two phases. The map phase works on user provided data and processes a <key,Value> pair. The reduce function collects the intermediate results of all individual computations on the chunks of data which is associated with the corresponding key values.

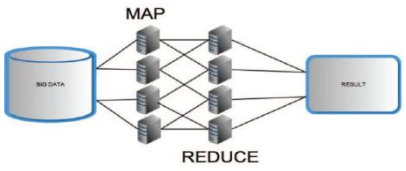

Figure 1: The MapReduce framework.

A single master job tracker manages all the computations and functionality that happens inside the MapReduce operation. There are multiple task trackers. Master is responsible for partitioning of input data, scheduling tasks , scheduling tasks and also reassigning failed tasks. It also manages inter machine communication and also keeps track of all the individual tasks. A file system is used to store input and output data. The very point that this has a single job tracker allows for the possibility of single point failure which means the failure of the system as a whole.

5)ALGORITHMIC DESIGN OF k-NN ALGORITHM WITH MAPREDUCE.

Now let's see the implementation of the K-NN algorithm, but before that the input and output of the MapReduce which is in key and value form needs to be handled.

There are two routines involved in this process. The Map routine calculates and lists out the distance of each data point and its corresponding classes. The reduce routine chooses the 'k' neighbours having the highest distance count, conducts the majority vote amongst them and resets the data point and class label along with the majority vote count. The input and output data needs to store in an organized manner and for these directories are used. Directories are created so that they that hold data vectors, trainfile to hold the training data and testfile to hold the testing data.

A. Algorithm for the Map Function

Algorithm 1 Mapper design for k-NN

0: **procedure** K-NN MAPDESIGN
0: Create list to maintain data points in the testing data-set
0: $testList = new\ testList$
0: Load file containing testing data-set
0: $load\ testFile$
0: Update list with data points from file
0: $testList <= testFile$
0: Open file containing training data set
0: $OPEN\ trainFile$
0: Load training data points one at a time and compute distance with every testing data point
0: $distance\ (trainData, testData)$
0: Write the distance of test data points from all the training data points with their respective class labels in ascending order of distances
0: $testFile <= testData(dist, label)$
0: Call Reducer
0: **end procedure**=0

B. Algorithm for the Reduce function

Algorithm 2 Reducer design for k-NN

0: **procedure** K-NN REDUCEDESIGN
0: Load the value of 'k'
0: Load testFile
0: $OPEN\ testFile$
0: Load test data points one at a time
0: $READ\ testDataPoint$
0: Initialize counters for all class labels
0: $SET\ counters\ to\ ZERO$
0: Look through top 'k' distance for the respective test data point and increment the corresponding class label counter
0: $for\ i = 0\ to\ k$
0: $COUNTER_i++$
0: Assign the class label with the highest count for the testDataPoint in question
0: $testDataPoint = classLabel(COUNTER_{max})$
0: Update output file with classified test data point
0: $outFile = outFile + testDataPoint$
0: **end procedure**=0

C. Algorithm to Implement the Map and Reduce Functions

Algorithm 3 Implementing kNN Function

0: **procedure** KNN FUNCTION
0: Read the value of 'k'
0: $SET\ 'k'$
0: Set paths for training and testing data directories
0: $SET\ trainFile$
0: $SET\ testFile$
0: Create new JOB
0: SET MAPPER to map class defined
0: SET REDUCER to reduce class define
0: Set paths for output directory
0: SUBMIT JOB
0: **end procedure**=0

5) ALGORITHMIC DESIGN OF FIUT ALGORITHM

FiDoop can be looked up to see the similarities and how it incorporates and implements the MapReduce process. One of the first challenge that will be faced is the way in which a serial algorithm like the FIUT gets parallelized. After j-itemsets are generated in the first phase a repetitive process is continuously running to construct k-FIU trees and to identify frequent k-itemsets until the k-value is from M to 2. Thus it is inferable that these tasks are carried out in a sequential manner. Now this sequential process must be converted so

that it is able to run on a distributed environment. This will be done as follows.

- The first phase of the FIUT involves two rounds of scanning the database is implemented in the form of two MapReduce jobs. The first job handles the first round of scanning and produces frequent itemsets. The second job handles the second round of scanning and produces k-itemsets by removing infrequent items in each transaction.
- Second phase of FIUT involving the construction of k-FIU tree and discovery of frequent itemsets is handled by a third MapReduce job in which j-itemsets are decomposed into a list (j-1)-itemsets,(j-2)-itemsets,..., and two-itemsets. In this job, the generation of shorter itemsets is independent of the longer itemsets.

The first MapReduce job discovers all frequent items. In this phase, the input given to the Map tasks is a dataset, and the resultant output of the Reduce tasks is all frequent one-itemsets. The second MapReduce job scans the database to generate *k*-itemsets by removing infrequent items in each transaction. The last MapReduce job the most complicated one of the threeconstructs *k*-FIU-tree and mines all frequent *k*-itemsets.
Below are the detailed descriptions of the FIUT algorithm and the various MapReduce jobs that happen in it.

A. FIUT Algorithm

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 ≤ 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
```

B. The First MapReduce Job

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
```

C. The Second MapReduce Job

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
```

6) CONCLUSION

The intention of combining the two algorithms is from the basic fact that they work very well in conjunction with the MapReduce paradigm. They are seamlessly integrated and work exceptionally on the distributed file systems. The k-NN algorithms pre computation provides for a very specific set of data points and when it is observed in the Big data

environment, this set derived from the k-NN algorithm is pretty large. Further implementing the FIU-tree algorithm on the same provides for a much more accurate result. Thus this has achieved the accuracy and reliability by gravely compromising time and memory constraints. With the future firmly set on the Big Data platform, it is time to look into more promising way of extracting or mining data that is reliable true to the problem and as always efficient.

REFERENCES

[1] M. J. Zaki, "Parallel and distributed association mining: A survey," *IEEEConcurrency*, vol. 7, no. 4, pp. 14–25, Oct./Dec. 1999.
[2] I. Pramudiono and M. Kitsuregawa, "FP-tax: Tree structure based gen-eralized association rule mining," in *Proc. 9th ACM SIGMOD WorkshopRes. Issues Data Min. Knowl. Disc.*, Paris, France, 2004, pp. 60–63.
[3] 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.
[4] 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.
[5] S. Kotsiantis and D. Kanellopoulos, "Association rules mining: A recent overview," *GESTS Int. Trans. Comput. Sci. Eng.*, vol. 32, no. 1, pp. 71–82, 2006.
[6] R. Agrawal and J. C. Shafer, "Parallel mining of association rules," *IEEETrans. Knowl. Data Eng.*, vol. 8, no. 6, pp. 962–969, Dec. 1996.
[7] The k-Nearest Neighbor Algorithm Using MapReduce Paradigm.
[8] Apache Hadoop. http://hadoop.apache.org/
[9] J. Venner, Pro Hadoop. Apress, June 22, 2009.

`