ISSN 2348-1196 (print)
International Journal of Computer Science and Information Technology Research ISSN 2348-120X (online)
Vol. 3, Issue 2, pp: (791-795), Month: April - June 2015, Available at: www.researchpublish.com

Finding Frequently Occuring Itemset Pair On Big Data

[1]Raju K Gupta, [2]Maheshwari R Tegampure, [3]Purushottam K Singh, [4]Shivani, [5]Akshay Kumar, [6]Prof. Prajakta Ugale

[1,2,3,4,5,6] MIT Academy of Engineering, Alandi

Abstract: **The frequent itemset mining (FIM) is one of the most important techniques to extract knowledge from data in many real-world application. The total data that we have today (on internet) is increasing day by day. But with this increase in data, we are also facing a big problem of extracting information from those data. Now an important thing is to extract information from that huge amount of data. This process of extracting information from the given data is also called as data mining. Though there were a number of methods (including parallel programming) to do so. But when they are applied to Big Data, they couldn't do too good. Apart these methods had their own counter effects such as 1) Balanced data distribution, and 2) Inter-communication costs. In this project, we investigate the applicability of FIM techniques on the MapReduce platform. Here in this project we are introducing two new methods for mining large datasets 1) FIC Algorithm (Fastest itemset calculating) (it focuses on speed) 2) Ec-Apriori Algorithm (It is optimized to run on really large datasets).**

Keywords: **bigdata, mapreduce, hadoop, frequent itemset.**

1. INTRODUCTION

Progress in digital data acquisition and storage technology has resulted in the growth of huge databases. This growth has occurred in all areas of human effort, from mundane (such as credit card usage records, supermarket transaction data, telephone call details) to the more exotic (such as images of astronomical bodies, molecular databases, and medical reports)[5]. These huge amount of data is generally called as Big Data. They came with some problem such as how to analyze it efficiently. It gave us an opportunity to find insight in new and emerging types of data and content, to make the business more business more quick and well-coordinated and to answer questions that were previously considered beyond our reach. But we are always interested in frequently occurring events. So, an event is interesting if it occurs frequently in the data. FIM tries to extract information from databases based on those frequently occurring events according to a user given minimum frequency threshold [2]. In this project, we have developed a system to analyse "Frequent Itemset Mining for Big Data" using Hadoop framework an open source distributed file system and map-reduce implementation. We will be moving the large size data onto Hadoop Distributed File System (HDFS). Map-Reduce algorithm will parse these log files.

The main objective of frequent itemset mining system is to extract valuable information from the given input data. When we use this information for creating analytical model provides best way to decision making. Also the advantages provided by the Hadoop technology with HDFS as distributed file system and Map reduce implementation for large-scale data processing in distributed cluster are the primary motivation for this project [5]. By taking the advantages of these technologies we can design the system for large scale data analysis with enhanced performance and scalability.

2. LITERATURE SURVEY

Since recent developments (in technology, science, user habits, businesses, etc.) gave rise to huge production and storage of those massive amounts of data. Hence the intelligent analysis of big data has become more important for both businesses and academics. Already from the start, Frequent Itemset Mining (FIM) has been an essential part of data

analysis and data mining. FIM tries to extract information from databases based on frequently occurring events. An event, or a set of events is interesting if it occurs frequently in the data, according to a user given minimum frequency threshold [1]. Applying frequent itemset mining to large databases is problematic:-1) very large databases do not fit into main memory (sol: Apriori Algorithm) 2) current approaches tend to keep the output and runtime under control by increasing the minimum threshold frequency [1]. Parallel programming is becoming a necessity to deal with the massive amounts of data, which is produced and consumed more and more everyday. Parallel programming architecture algorithms are classified into two types:- Shared and Distributed memory. On shared memory systems, all processing units can concurrently access a shared memory area [4]. On the other hand, Distributed systems are composed of processors that have their own internal memories and communicate with each other by passing messages. MapReduce framework proposed by Google, which simplifies the programming for distributed data processing, and the Hadoop implementation by Apache Foundation, which makes the frame work freely available for everyone [5]. So distributed programming is becoming more and more popular. The initial design principles of the MapReduce framework do not fit well for the FIM (frequent itemset mining) problem. It became necessity to provide MapReduce with efficient and easy to use data mining methods. Considering its availability and wide spread usage in industry we introduce two algorithms that exploit the MapReduce framework to deal with two aspects of the challenges of

FIM on Big Data:-

1) FIC Algorithm

2) Ec-Apriori

3. EXISTING ALGORITHMS

Data mining requires parallel methods. There are limited algorithms for adapting map reduce framework. These algorithms do the counting steps in parallel. The first one is single pass counting which is used to check for each candidate generation and steps to count frequency utilizing map reduce. The second one is fixed pass combined counting which is used to start candidate with different lengths, phases and counts their frequencies. The third one is dynamic pass counting where n is for different length and p for different phases both are calculated dynamically by number of candidates [1].

The Apriori method is similar to single pass counting with little difference. Even MApriori is used to mine frequent itemsets between vertical and horizontal database. All the levels structure of Apriori is not usable or implicable because of high overhead of mapreduce cycles. Breadth first search and apriori produces short frequent itemsets as it cannot handle long one because of combinatorial explosion. Now FP-growths parallel version is parallel FP-growth. fp growth is frequent parallel growth[1].

The items are grouped in parallel frequent pattern and are distributed to mappers. Each mappers mines an independent fp-tree. To balance the groups of pfp frequencies of frequent items are used. The grouping strategy is not correct in pfp neither in speed nor in memory terms. Big data is restricted but is possible for some nodes in database memory. Big data says to balance the distribution for quick result we can use it as single as dividing the search space is not useful. FIM methods that use k- methods to cluster transactions to represent itemsets. The PARMA algorithm finds exact number of frequent itemsets and it guarantees that these itemsets will have great quality after finding results. To improve the applicability of map reduce framework some work needs to be done [1].

Twister and Nimble can be the examples. Twister advances the performance and nimble gives better tools. Core map reduce framework implementation should be focused. There are not many ways to mine frequent itemsets based on map reduce framework. PFP is good but it has some disadvantages. There are some problems with PFP such as scalability problem.

A) Search Space Distribution:

The most crucial part in accepting an algorithm is limitation in communication of tasks that are running in parallel. The tasks should be started initially so that the nodes do not interact with each other during processing. The prefix tree needs to be divided into independent groups. Each group should be mined differently on different devices. The processing time of an algorithm is dependent on processing time of longest tasks. Balancing the running time is important for managing total time computation.

Eclat says that execution time is a part of frequent itemsets during division. However these have some limitations to visualise the time taken to calculate a sub-tree. Total itemsets is bad for determining division because of its monotonic property. This property prunes the search space that is reduces the size of decision tree by removing section of the tree [2]. Thus large portions have to be created and determined. Assigning prefixes to the nodes provides better balancing of load. Benefit of such technique is not as proper as round robin.

The most crucial part for developing a balanced division is the size of prefixes. Shards need to be divided among various nodes. Dividing the tree from lower depth gives better balancing. This behaviour should be there because longer FIs gives more information about data and also divides the tree more accurately or precisely.

Our accepted algorithms shows interdependence of sub trees and utilizes longer FIs as prefixes for giving better balancing of loads. Both the algorithm mines the itemsets parallely. They mine the itemsets upto a certain length to calculate k-length prefixes. Where Pk=P1,P2,....Pn.

Here Pk is divided into m groups. Where m is distributed nodes.

Prefixes of each group is passed to worker node to be used as database based on conditions. Prefix of each part of the tree is unique and it does not depend on other parts of trees. Thus each node works without depending on other nodes or even not overlapping. Frequent itemsets do not require post-execution.

Prefix trees are developed and are sorted by frequency in ascending order. I=i1,i2,....in. Here support of ia should be less than or equal to support of ib ⇔a<b

This way of arranging helps to prune the prefix tree at lower sides and gives lesser run time. Further details are explained as follows.

B) Proposed Methods:

Frequent Item Set Mining is a method for market based analysis. It aims at finding regularities in the shopping behaviour of customers of supermarkets, mail order of companies or on-line shops etc. More specifically it can be said the methods of finding set of products that are frequently bought together.

Finding Frequent Itemset using mapreduce:

Now we know the importance of Parallel programming and MapReduce, so using these two concepts we propose two new methods for mining big data where threshold frequency can be kept low. The methods are FIC Algorithm which is a pure Eclat method and it is generally used when the quantity of data is low and the processing time is important. This method is good until we don't have to deal with huge amount of data.

So for the case when we have to deal with what we call it as Big data then we introduce a new hybrid methods called as Ec-Apriori Algorithm. This method is a hybrid of Apriori and FIC Algorithm. This method is usually for large databases where it does not completely fits into the memory.

Method 1:

A) FIC Algorithm:
This algorithm is basically an ECLAT method. ECLAT is Equivalence Class Clustering and bottom up Lattice Traversal. It's a method for frequent itemset generation. It searches in DFS manner. FIC Algorithm is a distributed version of Eclat. It distributes the search space more evenly among different processing units. Earlier methods such as Partition algorithm, which oftenly worked on the methods of distributing the database or workload into n numbers of equally sized sub databases [1]. These sub database is also called as Shards. Then these shards are mined separately to find the frequent item and then these results which are locally frequent are combined together and processed again to prune the globally infrequent items. So we can say that this method is an expensive one. apart from this it has large communication cost too, as the number of sets that is to be mined can be very large, hence the set that will be recounted will be also very large. Such sort of algorithms are generally avoided in Hadoop. But the method that we will implement will be free from such an anomalies as we will divide the search space rather than the data space. Hence this method will reduce the communication cost as no extra communication is required between the mappers and no checking for overlapping mining results has to be done. Concluding we can say that for mining large dataset, memory-wise is the best fit for this method FIC Algorithm i.e. Eclat using diffsets. We divide the operation of this method in 3 parts and in order to maximally benefit the cluster environment we distribute all the three parts among multiple mappers. And for this we utilize the vertical database format.

ISSN 2348-1196 (print)
International Journal of Computer Science and Information Technology Research ISSN 2348-120X (online)
Vol. 3, Issue 2, pp: (791-795), Month: April - June 2015, Available at: www.researchpublish.com

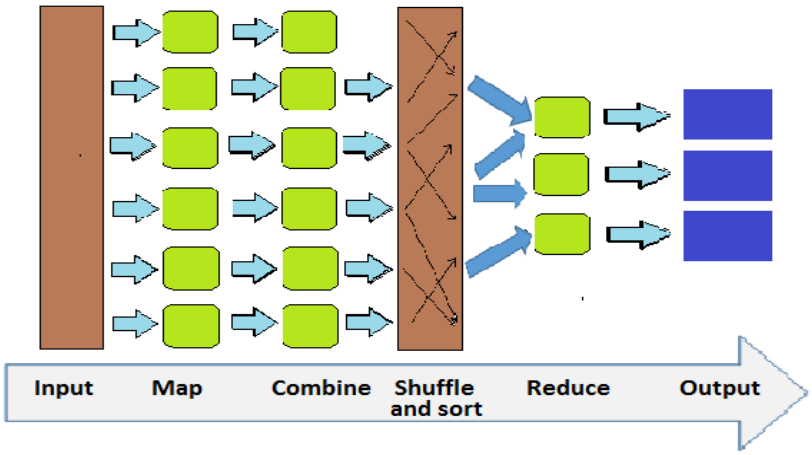

Fig.:- MapReduce using Eclat

Steps are as followed:

1) Finding the local Frequent Items: In this step, the vertical database is partitioned into sub databases and then divided to the separate mappers available. The mapper then removes each of the limited items from the sub databases. Then in the reduced phase, the frequent items are collected.

2) k-FIs Generate: In this step, the set of limited itemsets of k-size are obtained. Firstly, the frequent items are divided amongst the m mappers. Each mapper separately combines the items. Then the reducer assigns these items to another batch of m mappers by using round robin method.

3) Mining the Subtree: In this step the prefix tree is mined. Here the prefix tree is used a s the big data. The prefix tree is a structure which describes an itemset, the exact path from the root to the node. The tree is divided into separate groups. Each group is then mined individually on different mappers.

Method 2:

***A) Ec-Apriori Algorithm*:**

Our previous method (***FIC Algorithm***) holds good as far as our database is small i.e. can fit into memory. So we thought of a new algorithm called as Ec-Apriori Algorithm which is basically a hybrid method. It combines principles from both Apriori and Eclat method. Ec-Apriori Algorithm employs Apriori algorithm to find relatively shorter frequent itemsets until their list of transactions can fit into the memory. Then, it switches back to Eclat algorithm to distribute the search space and to complete frequent itemset mining. If your database is small and you need Hadoop just for a speed up, you can use FIC Algorithm for fast mining [1]. If your data is larger than available memory of the nodes, then Ec-Apriori Algorithm is your only option. Mind that, Ec-Apriori Algorithm can be used for databases with any size.

Again this method is implemented in three following steps:-

1) **Generation of k-FIs:** To solve the problem of large tid-lists, k-FIs are generated using breath first search. This can be done by giving different mappers a part of database and report items for which we need to know support. This can be viewed as word counting problem (i.e complete document is divided into parts and every mapper get a part and word is reported for which count is to be known).Then reducer combine all frequencies and give only the itemsets that are globally frequent. Then itemsets report by reducer are again distributed to different mappers for next BFS step, the above steps should be repeated to obtain k-FIs set. In Apriori technique keeping candidate in memory might be possible for few levels but when candidate set is large enough to fit in memory, then it should be partition among mappers.

2) **Finding of Potential Extensions:** After the k-FIs are generated, next step is to find extensions i.e tid-lists for (k+1)-FIs are to be obtained. This can be obtained similar to word counting approach as discussed above. But we will report local tid-lists and tid-lists from mappers are combined by reducer to one global list and assigns groups to mappers.

3) **Mining the Subtree:** Final step is subtree mining, Here mappers work on different prefix group (A prefix group is a database that fits into memory).Mining takes place on database (prefix group) for frequent itemsets using DFS.

4. CONCLUSION

Here in this paper we will be working on for mining BigData as to extract information about frequently occurring itemsets. Although a number of methods are already existing but with some flaws. So we hereby present two methods namely FIC Algorithm and Ec-Apriori Algorithm which will solve our existing problems and give proper desired output by generation k-FIs. If the processing speed is important and the amount of data is less then we will use our first method called FIC Algorithm and if our data is too large in size and processing is more important than the time taken then for such cases we will be working on our second method called as Ec-Apriori Algorithm. So, we will be working on these two algorithms to eradicate the current flaws in the existing method.

Apart from this we will be implementing this algorithm on single node as well as on the cluster. So here we will be showing the importance of using clustered environment by the processing time taken by them, as we are expecting that as the number of nodes in the cluster will increase the time taken by the system will reduce.

REFERENCES

[1] Sandy Moens, Emin Aksehirli and Bart Goethal. Frequent Itemset Mining for Big Data. IEEE Big Data, 2013 IEEE International, pages 111-118, 2013.

[2] Christian Borgelt. Frequent Pattern Mining. Intelligent Data Analysis and Graphical Models Research Unit, European Centre for Soft Computing, Ppt.

[3] Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. IEEE Operating Systems Design & Implementation, pages 0-10.

[4] Tom White. Hadoop: The Definitive Guide, O'REALLY publication, 2012.

[5] PAUL ZIKOPOULOS, Chris Eaton, Thomas Deutsch, George Lapis, Chris Eaton. Understanding Big Data: Analytics for Enterprise Class Hadoop and Streaming Data.