INTERNATIONAL JOURNAL OF INNOVATIVE TECHNOLOGY AND CREATIVE ENGINEERING (ISSN:2045-8711)
VOL.6 NO.12 DECEMBER 2016, IMPACT FACTOR: 0.61

An Adaptive Intelligence Technique for Ultra Metric Tree Frequent Item Set Mining

K.Mohankumar
M.Phil Research Scholar, Department of Computer Science,
Nandha Arts and Science College
Erode, Tamil Nadu, India.
Mrs.M.Santhoshmani
Assistant Professor, Department of Computer Science,
Nandha Arts and Science College
Erode, Tamil Nadu, India.
Email:shrisanu21@gmail.com
Dr.S.Prasath
Assistant Professor, Department of Computer Science,
Nandha Arts and Science College
Erode, Tamil Nadu, India.
Email: softprasaths@gmail.com

Abstract-Frequent items are an item that occurs frequently in the dataset. Frequent item set mining (FIM) is a one of the core data mining operation. Frequent item set mining is mainly used for market basket analysis. Consider an example a set of items that contains bread and butter which always occurs frequently together. A traditional frequent item set mining algorithm are Apriori and FP-growth algorithm. Apriori algorithm is a level-wise iterative approach were k items are used to generate the k+1 items. Apriori algorithm consists of two steps join step and prune step. Initially candidate items are generated by joining process after that by checking the minimum support count frequent items will be generated. The process will be repeated until all k frequent items generation. However, it has a disadvantage that many candidate items should generate which increases the computing time. To overcome that a pattern growth approach algorithm is proposed which significantly reduce the size of candidate sets. FP-Growth algorithm adopts a divide and conquers strategy for finding frequent item sets. It also has some disadvantage that frequent items are generated by repeated scanning of database and recursive traversing of tree.

Keywords— Data Mining, Frequency Item Set, Apriori.

1. INTRODUCTION

Frequent items are an item that occurs frequently in the dataset. Frequent item set mining (FIM) is a one of the core data mining operation. Frequent item set mining is mainly used for market basket analysis. Consider an example a set of items that contains bread and butter which always occurs frequently together. A traditional frequent item set mining algorithms are Apriori and FP-growth algorithm. Apriori algorithm is a level-wise iterative approach were k items are used to generate the k+1 items. Apriori algorithm consists of two steps join step and prune step. Initially candidate items are generated by joining process after that by checking the minimum support count frequent items will be generated. The process will be repeated until all k frequent items generation. However, it has a disadvantage that many candidate items should generate which increases the computing time. To overcome that a pattern growth approach algorithm is proposed which significantly reduce the size of candidate sets. FP-Growth algorithm adopts a divide and conquers strategy for finding frequent item sets. It also has some disadvantage that frequent items are generated by repeated scanning of database and recursive traversing of tree.

1.1 Frequent Item set Ultra metric Tree Using Map Reduce

Now days, FIM is most importantly used by researchers because it is widely applied in real world to find the frequent item sets. As a volume of database increases day by day, the problems of scalability and efficiency become more severe. As a solution to this problem, we design a parallel mining of frequent item set using FIUT algorithm on Map Reduce framework. In this paper we incorporate enhanced Frequent Item set Ultra metric Tree (E-FIUT), rather than traditional FP-Tree. FIUT has major four advantages over traditional FP-tree like; it involves only two round of scanning which minimizes I/O overhead. Then the EFIUT is a highly improved way to partition a database, which considerably reduces the search space. Next is here only frequent items in each transaction are inserted as nodes into the EFIUT for compressed storage. At last all frequent item sets are generated without traversing the tree recursively by checking the leaves of each EFIUT which significantly reduces computing time.

2. BIGDATA

In the 21st century, it is increasingly inseparable from the network, people visit dozens or even hundreds of pages, or upload photos or speech every day, which makes the data content on the network into a geometric growth, and the traditional technical architecture has become increasingly unable to meet the current needs of the vast amounts of data. Therefore, researching massive data processing and storage become more and more popular nowadays.

Big data is a large data that it becomes difficult to process the conventional database systems. If the data is very large, moves very fast, or doesn't fit the structures of the database architectures. To gain value from this data, choose another way to process the data. Big Data in general is defined as high volume, velocity and variety information assets that demand cost-effective, innovative forms of information processing for enhanced insight and decision making. Big Data is the frontier of the firm's ability to store, process and access large volume of data it needs to operate effectively, make decisions, reduce risks, and serve customers. However, the amount of data generated can often be very large for a single computer to process in a reasonable amount of time. Furthermore, the data itself may be too big to store on a single machine. Therefore, in order to reduce the time taken to process the data, and to allocate the storage space for large files, it is necessary to write programs that can execute on multiple computers and distribute the workload among them.

3. HADOOP

Hadoop is the foundation for biggest data architecture. Apache hadoop is an open source java programming framework for fast storing and fast processing large data sets with cluster of commodity hardware. Cluster is a set of machine in single LAN (Local Area Network). The Hadoop is mainly constituted by the underlying distributed file system HDFS (Hadoop Distributed File System) and Map Reduce layer of parallel programming model engine. Hadoop is used by various universities and companies like Google, eBay, Facebook, IBM, LinkedIn and Twitter. HDFS is a reliable distributed file system that provides high-throughput and scalable access to data. Map Reduce is a distributed framework for executing the work in parallel. Hadoop has the master/slave architecture for both processing and storage. HDFS is a specially designed file system for storing massive amount of data sets with cluster of commodity hardware with steaming access pattern. Steaming access pattern means write once and read any number of times but don't change content of files in file system. HDFS differ from other file system by its significant. HDFS is a very large distributed file system which is highly fault-tolerant, provides high throughput access to the large data and deployed on low-cost hardware.

HDFS is mainly used for storing data, and simply adding the number of servers can achieve growth in storage capacity and computing power. Map Reduce can make full use of the computing resources of each server's CPU, which efficiently handles with the stored data and calculations. To address the above issues, Google developed the Google File System (GFS), which is a distributed file system architecture model for processing large amount of data and created the Map Reduce programming model. The Map Reduce programming model is for processing the massive amount of data in parallel. Hadoop is an open source software which mange Map Reduce framework, written in Java, originally developed by Yahoo.

A Map Reduce consists of two tasks namely the Map and Reduce task. Each Map task takes key-value pair as input and produce key-value pair as an output. The input data are split into various input splits. Based on the number of input splits Mapper will be assign. Record Reader is an interface between input split and Mapper which is used to convert record into key value pair. Mapper will read key value pair as an input and produce key value pair as an output. Now the Reducer will combine all the intermediate values associated with a particular key. Both input pairs of Mapper and Reducer are managed by the HDFS. The advantage of Map Reduce is highly scalable, transparent fault-tolerant processing and automatic parallelization.

4. EXISTING SYSTEM

In Existing System Rather than considering Apriori and FP-growth, we incorporate the frequent items ultra metric tree (FIU-tree) in the design of our parallel FIM technique. We focus on FIU-tree because of its four salient advantages, which include reducing I/O overhead, offering a natural way of partitioning a dataset, compressed storage, and averting recursively traverse. Parallel algorithms lack a mechanism that enables automatic parallelization, load balancing, data distribution, and fault tolerance on large computing clusters. Not efficient, require more time for mining. Existing parallel mining algorithms for frequent item sets lack a mechanism that enables automatic parallelization, load balancing, data distribution, and fault tolerance on large clusters. As a solution to this problem, we design a parallel frequent item sets mining algorithm called FiDoop using the Map Reduce programming model. To achieve compressed storage and avoid building conditional pattern bases, FiDoop incorporates the frequent items ultra metric tree, rather than conventional FP trees. In FiDoop, three Map Reduce jobs are implemented to complete the mining task. In the crucial third Map Reduce job, the mappers independently decompose item sets, the reducers perform combination operations by constructing small ultra metric trees, and the actual mining of these trees separately. We implement FiDoop on our in-house Hadoop cluster. We show that FiDoop on the cluster is sensitive to data distribution and dimensions, because item sets with different lengths have different decomposition and construction costs. To improve FiDoop's performance, we develop a workload balance metric to measure load balance across the cluster's computing nodes. We develop FiDoop-HD, an extension of FiDoop, to speed up the mining performance for high-dimensional data analysis. Extensive experiments using real-world celestial spectral data demonstrate that our proposed solution is efficient and scalable.

5. PROPOSED SYSTEM

The proposed system a new data partitioning method to well balance computing load among the cluster nodes; we develop FiDoop-HD, an extension of FiDoop, to meet the needs of high dimensional data processing.

5.1 Frequent Item set Ultra metric Tree

I-FIUT is a new method for mining frequent item sets from the database. I-FIUT has four major advantages over traditional FP-tree like: it involves only two round of scanning which minimizes I/O overhead. Then the I-FIUT is a highly improved way to partition a database, which considerably reduces the search space. Next is here only frequent items in

each transaction are inserted as nodes into the I-FIUT for compressed storage. At last all frequent item sets are generated without traversing the tree recursively by checking the leaves of each I-FIUT which reduces computing time significantly. I-FIUT consists of two phases to generate the frequent item sets from the transactions by two rounds of scanning the database.

5.2 Generating one Item sets and K Item sets

Phase1 consists of two round of scanning the database. At the first round of scanning the database frequent one item will be generated based on the minimum support count. At the second round of scanning the database all k-items will be generated by pruning the infrequent items from each transaction.

5.3 Generating Frequent K Item sets

Phase2 consists of a two process decompose each 'h' item sets into 'k' item sets. After decomposing process, the repetitive construction of K-FIU-Tree and all 'k' frequent item sets are generated by checking the leaves of FIU-Tree where 'k' is from M down to 2. After decomposing process 'k' item sets are generated that are used for the construction of K FIU Tree. Initially the root is labeled as null. Then each 'k' item sets are inserted into the tree. If first frequent item exists as one of the children of the root, then it denotes the child as a temporary 1st root, if it is not existing then add a new node for this item as a child of the root node and denote it as temporary 1st root. Then the sth frequent item of the k item set, where 's' is from 2 to k - 1, check if the sth frequent item exists as one of the children of the temporary (s-1)th root, then denote the child as a temporary sth root. If it does not exist, then add a new node for this item as a child of the temporary (s-1)th root and denote it as a temporary sth root. This process is repeated until K-FIU Tree is constructed. By checking the leaf node all k frequent items will be generated.

5.4 I-FIUT

Each phase of Frequent Item set Ultra metric Tree is explained with an example. Consider the 5 transactional database D as shown in the Table.5.1

Table 5.1 Database D

TID	ITEMS BOUGHT
100	a,c,d,f,g,i,m,p
200	a,b,c,f,l,m,o
300	b,f,h,j,o
400	b,c,k,s,p
500	a,c,e,f,l,m,n,p

During the phase 1 at the first round of scanning the database frequent one item sets will be generated with the minimum support count value 2. Table 5.2 shows the frequent one item set of the database D.

Table 5.2 Frequent 1 Item sets

A	3
B	4
C	5
F	4
M	3
P	4

During the phase 1 at the second round of scanning the database all 'k' item sets will be generated by pruning the each infrequent item from each transactional datasets. Table 5.3 shows all 'k' item sets.

Table 5.3 All K Item sets

1-itemsets	a,c,f,m,p
5-itemsets	a,b,c,f,m
4-itemsets	Ø
3-itemsets	b,c,p
2-itemsets	b,f

5.5 Parallel Mining of Frequent Item sets using I-FIUT on Map reduce Framework

As a volume of database increases day by day traditional frequent item set mining algorithms becomes inefficient. As a solution to this problem parallel mining of frequent item sets using I-FIUT algorithm is implemented on Map Reduce framework. Here we using I-FIUT algorithm rather than traditional FP-Tree algorithm because to avoid building conditional patterns and to achieve compressed storage. We build this using Hadoop framework. The working flow of I-FIUT algorithm on Map Reduce framework consists of three Map Reduce job. Synthetic datasets are used for the experimental analysis. The working flow of I-FIUT based Map Reduce framework. It consists of three Map Reduce job. The output of the first Map Reduce is all frequent one item sets. The second Map Reduce job is responsible for creating all k item sets. Finally the third Map Reduce job is responsible for creating all frequent k item sets.

5.6 Frequent One Item sets Generation

The first Map Reduce job is responsible for mining all frequent one-item sets. A transaction database is partitioned into multiple input split files stored by the HDFS across multiple data nodes of a Hadoop cluster. Number of mapper will be executed based on number of input split. Each mapper sequentially reads each transaction from its local input split, where each transaction is stored in the format of key value pair<Long Writable offset, Text record> by the record reader. Then, mappers compute the frequencies of items and generate local one-item sets. Next, these one-item sets with the same key emitted by different mappers are sorted and merged in a specific reducer, which further produces global one item sets. Finally, infrequent items are pruned by applying the min support and consequently, global frequent one-item sets are generated and written in the form of pair<Text item, Long Writable count> as the output from the first Map Reduce job. Importantly, frequent one-item sets along with their counts are stored in a local file system, which becomes the input of the second Map Reduce job.

5.7 All K Item sets Generation

Given frequent one-item sets generated by the first Map Reduce job, the second Map Reduce job applies a second round of scanning on the database to prune infrequent items from each transaction record. The second job marks an item set as a k-item set if it contains k frequent items ($2 \leq k \leq M$, where M is the maximal value of k in the pruned transactions). Each mapper of the second job takes transactions as input. Then, the mapper emits a set of pair <Array Writable item sets, Long writable ONE>, in which item sets is composed of the number of the items produced by pruning and the set of items. These pairs obtained by the second Map Reduce job's mappers are combined and shuffled for the second job's reducers. After performing the combination operation, each reducer emits key/value pairs, where the key is the number of each item set and the value is each item set and its count. More formally, the output of the second Map Reduce job is pair<Int Writable item number, Map Writable<Array Writable k-item, Long Writable

5.8 Frequent K Item sets Generation

The third Map Reduce job a computationally expensive phase is dedicated to: 1) decomposing item sets; 2) constructing k-FIU trees and 3) mining frequent item sets. The main goal of each mapper is twofold: 1) to decompose each k-item set obtained by the second Map Reduce job into a list of small-sized sets, where the number of each set is anywhere between 2 to k − 1 and 2) to construct an FIU-tree by merging local decomposition results with the same length. The third Map Reduce job is highly scalable, because the decomposition procedure of each mapper is independent of the other mappers. In other words, the multiple mappers can perform the decomposition process in parallel. Such an FIU-tree construction improves data storage efficiency and I/O performance; the improvement is made possible thanks to merging the same item sets in advance using small FIU trees. The Map function of the third job generates a set of key/value pairs, in which the key is the number of items in an item set and the value is an FIU-tree that is comprised of non leaf and leaf nodes. Non leaf nodes include item-name and node-link, leaf nodes include item-name and its support. In doing so, item sets with the same number of items are delivered to a single reducer. By parsing the key-value pair (k2, v2), the reducer is responsible for constructing k2-FIU-tree and mining all frequent item sets only by checking the count value of each leaf in the k2-FIU-tree without repeatedly traversing the tree. Figure 6.4 illustrates the Map and Reduce functions. Here, the details on the function of t-FIU-tree generation (t-item set) can be found. The decompose () function is a recursive one, decomposing an h-item set into a list of k-item sets, where k is an integer between 2 and h.

6. PEFORMANCE EVALUATION

The performance for proposed methods can be evaluated by using the following parameters. Parameters which are considered for evaluating the experiments are:

6.1 Minimum Support Count

Minimum support count plays the important role in mining frequent item sets. When we increase the minimum support threshold the running time of the proposed algorithm reduces. A small minimum support slows down the performance of the evaluated algorithms. This is because an increasing number of items satisfy the small minimum support when the min support is decreased; it takes an increased amount of time to process the large number of items. Figure 6.1 shows the execution time of four different minimum support counts.

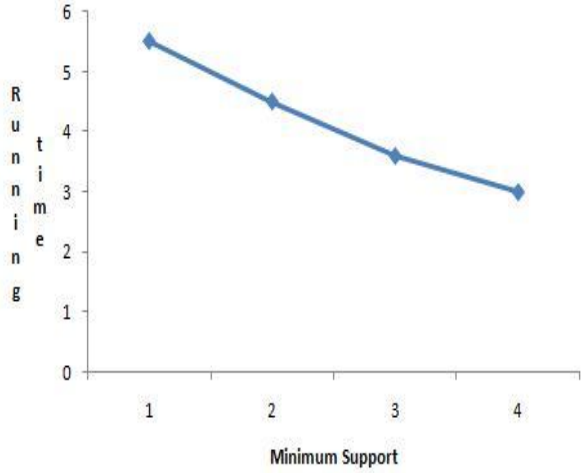

Fig.6.1 Minimum Support

6.2 Scalability

In this experiment, evaluate the scalability of the proposed algorithm when the size of input dataset grows dramatically. The parallel mining process is slowed down by the excessive data amount that has to be scanned twice. The increased dataset leads to a long scanning time. An output of the second Map Reduce job are distributed and stored in intermediate files based on the length of item set these files are accessed by the third Map Reduce job as an input. Further, the decomposed results are written into these external files. The scalability of the proposed algorithm is higher when it comes to parallel mining of an enormous amount of data.

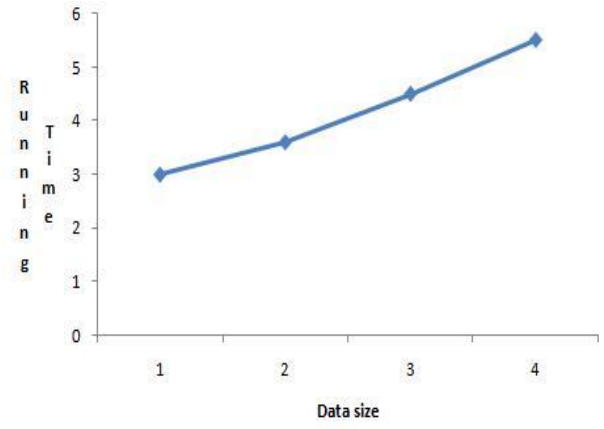

Fig.6.2 Scalability

7. CONCLUSION

To solve the scalability and efficiency in the existing parallel mining algorithms for frequent item sets for frequent item sets, applied the parallel mining of frequent item sets using Frequent Item set Ultra metric Tree on Map Reduce framework. We incorporate the Frequent Item set Ultra metric Tree rather than conventional FP trees, thereby achieving compressed storage and avoiding the necessity to build conditional pattern bases. The proposed algorithm integrates three Map Reduce jobs to accomplish parallel mining of frequent item sets. At the end of the third Map Reduce job all frequent K item sets are generated. To evaluate the performance of the proposed I-FIUT algorithm on Map Reduce framework we use synthetic datasets in our experiments.

REFERNCES

[1] Chang E.Y., Li H., Wang Y., Zhang D. and Zhang M. (2008), 'PFP: Parallel FP-growth for query recommendation', in Proc. ACM Conf. Recommend.Syst., Lausanne, Switzerland, pp. 107–114.

[2] Chang W.L., Chen P.L. and Lin K.W. (2011), 'A novel frequent pattern mining algorithm for very large databases in cloud computing environments', in Proc. IEEE Int. Conf. Granular Comput. (GrC), Kaohsiung, Taiwan, pp. 399–403.

[3] Chunyan H., Hong S., Huaxuan Z. and Shiping s. (2013), 'The study of improved FP-growth algorithm in MapReduce' in Proc. 1st Int. Workshop Cloud Comput. Inf. Security, Shanghai, China, 2013, pp. 250–253.

[4] Cong S, Han J., Hoeflinger J. and Padua D. (2005) 'A sampling-based framework for parallel data mining', in Proc. 10th ACM SIGPLAN Symp. Prin. Pract. Parallel Program., Chicago, IL, USA, pp. 255–265.

[5] Dean J. and Ghemawat S. (2008), 'MapReduce: Simplified data processing on large clusters', Commun. ACM, vol. 51, no. 1, pp. 107–113.

[6] Dean J. and Ghemawat S. (2010), 'MapReduce: A flexible data processing Tool', Commun. ACM, vol. 53, no. 1, pp. 72–77.

[7] Han E H., Karypis G. and Kumar V. (2000) 'Scalable parallel data mining for association rules' , IEEE Trans. Knowl. Data Eng., vol. 12, no. 3, pp. 337–352.

[8] Han J., Mao R., Pei J. and Yin Y. (2004), 'Mining frequent patterns without candidate generation: A frequent-pattern tree approach', Data Min. Knowl. Disc., vol. 8, no. 1, pp. 53–87.

[9] Hsueh S.C., Lin M.Y. and Lee P.Y. (2012), 'Apriori-based frequent itemset mining algorithms on MapReduce', in Proc. 6th Int. Conf. Ubiquit. Inf. Manage. Commun. (ICUIMC), Danang, Vietnam, pp. 76:1–76:8.

[10] Hsu T.J., Tsay J.Y. and Yu J.R. (2009), 'I-FIUT: A new method for mining frequent itemsets', Inf. Sci., vol. 179, no. 11, pp. 1724–1737.

[12] Kitsuregawa M. and Pramudiono I. (2003), 'Parallel FP-growth on PC cluster', in Advances in Knowledge Discovery and Data Mining. Berlin, Germany: Springer, pp. 467–473.