Frequent Pattern Mining for Stream Data by Using Hadoop GM-Tree and GTree

Than Htike Aung, Nang Saing Moon Kham
Information Science Department, University of Computer Studies, Yangon
thanhtikeaung@ucsy.edu.mm, moonkhamucsy@ucsy.edu.mm

Abstract

Since origination of mining, frequent pattern mining has become a mandatory issue in data mining. Transaction process for mining pattern needs efficient data structures and algorithms. This system proposed tree structure, called GMTree(Generate and Merge Tree)-GTree(Group Tree), which is a hybrid of prefix based incremental mining using canonical order tree and batch incrementing techniques. Proposed system make the tree structure more compact, canonically ordered of nodes and avoids sequential incrementing of transactions. It gives a scalable algorithm with minimum overheads of modifying the tree structure during update operations. It operates on extremely large transaction database in dynamic environment which is especially expected to give better results in this case.The proposed system used Apache Hadoop and hybrid GMTree-GTree. The results shows Hadoop implementation of algorithm performs more times better than in Java.

1. Introduction

Over the past few decades, many researchers have proposed many algorithms for discovering frequent item sets from a given data set[19,12,18,14,15]. Generally, the data set can be of two types static or dynamic. Most algorithms focus on a static data set. However, static data mining algorithms which can't be applied to a dynamic data set. Specially, a real time method is preferred to a batch processing method. Mining on streaming data can be categorized into the following types on the basis of the window concept: Landmark, damped and sliding-window. Landmark model, mainly focus on data set that is observed from a fix time in the past to present time. In the damped approach, frequent item set are extracted in data stream where every transaction of the data is allotted a weight that reduce with age. The sliding-window model, the item sets are collected in a certain interval of time from the present time.

The proposed system's algorithm are sliding-window technique [2],[12] which moves per unit batch. The proposed system efficiently representing the transaction makes use of base-tree that is constructed from GMTree, which is almost the same as CanTree. The change between the two is that when a new arrives, similar items generally form a single node in the new tree for comparison (not need each new item compared as a CanTree.) and which is not need restructure the entire tree same as FPTree.

GMTree node represents a set of nodes that have the same data item in the base tree(GMTree or parent GTree). That is the proposed algorithm called Hadoop GMTree-GTree. It is very sample and efficient. The algorithm has the following properties.

- Single database scan.
- Usage of sliding-window
- Similar items generally form a single node in the new tree for comparison (not need each new item compared.)
- Finding exact and complete frequent item sets.

2. Related Work

A data structure called FPTree which was used in FP-growth algorithm [11] to compactly represent a transaction DB into main memory. FP-growth algorithm produces frequent itemsets by a divide-and-conquer method, and needs only two DB scans. This algorithm shows very efficient memory usage and processing cost.

This successful data mining method has led to many FP-growth-based algorithms for a static data set. But, because of the two DB scans, FPTree based FP-growth algorithm cannot be applied to stream data mining.

However, FP-growth algorithm has used on many stream data mining methods, such as FP-stream [1], DSTree [16], CanTree [4,6], FUFP-Tree [21,22], CPSTree [20], and others [9,8]. Furthermore, there

are many studies based on apriori-based algorithms, such as SWF [6], SWFI stream [2], and MFI-TransSW [10]. These incremental mining methods have shown good performance and mining results for several applications, but basically which have a limitation in dealing with data streams.

As explore in several studies [7,17,20], considerably more processing cost and memory is commonly needed to generate and test the candidate itemsets. The results in huge processing cost for candidate itemsets generation, especially if there are a huge number of items (or the length of candidate itemsets is long).

3. Proposed Methodology

The proposed system used GMTree-GTree data structures with sliding-window method. This data structures are more efficient than GMTree-GTree data structures. This data structures of new tree for data nodes need compared on similar item generally form single node. In CanTree, it may generate a skewed tree with too many branches and hence with too many nodes. It considers one transaction from database at a time, which drops its time efficiency. It produces concise tree only if majority of transactions have a common pattern.

3.1. GM-Tree

This section describe tree structure called GM-Tree for maintaining frequent patterns found in a dynamic database with an improved functionality and performance by combining the prefix based incremental mining using canonical ordering and batch incrementing techniques. Incremental mining technique is proposed for the maintenance of frequent item sets that are discovered in transaction databases. It updates the frequent patterns very efficiently when databases are frequently changed by additions, deletions and modifications of transactions. Batch Incremental technique refers to the merging of two dataset (here in the form of a tree) to obtain a new dataset that is equivalent to the entire database formed by the two sets. Combination of these two approaches helps to make our tree structure as follows:

(1) More compact.
(2) Canonically ordered of nodes.
(3) Avoids sequential incrementing of transactions.
(4) Gives a scalable algorithm with minimum overheads of modifying the tree structure during update operations.

The proposed algorithm makes a single scan through the initial database to construct a tree structure. The tree so formed has items arranged from root to the leaves in a lexicographic manner, hence ordering of the items are unaffected by their frequencies. The support frequency is taken into account while mining the tree. Now to deal with the dynamic environment, a new similar tree is constructed from new transactions in the database. Once created, the new tree is merged with the last updated tree forming the corresponding tree structure for the whole updated database, avoiding a re-scan of the entire updated database and thus providing a better efficiency. The above statements can be summarized into two important properties of the GM-Tree described below:

Property 1: Nodes in GM-Tree are ordered lexicographically and thus the ordering is unaffected by changing item frequency.

Property 2: New transactions are used to generate another tree which is then merged with the last updated tree, preventing re-scan of the entire updated database.

Figure1 shows the GM-Tree generated from Table 1.

In summary, our proposed GM-Tree solves the limitations of the above mentioned trees as follows:

In case of the GM-Tree, nodes are arranged in a canonical (i.e. lexicographic) ordered and hence while merging two trees, we do not require to check and swap nodes with the changing frequency.

GM-Tree is not affected by frequency count and thus swapping or bubbling and re-scanning of the nodes can be completely avoided which makes it more time efficient.

During GM-Tree construction, when a new tree is formed, only the nodes of this new tree needs to be compared with the last updated tree nodes. Thus, when the data size is increased and the size of the tree increases, a large number of unnecessary comparisons can be avoided. This indicates that GM-Tree is well suited for extremely large database.

GM-Tree needs more memory while merging two trees but reduces the computational time drastically as there is no swapping or re-scanning of nodes required. Moreover, in this modern world, space requirement (i.e. main memory) is no more a big concern [25][16]. Algorithm steps for GMTrees areas follow:

(1) Create a Tree (T) from the content of the original database ($db_{original}$) and order the nodes of T lexicographically.
(2) Create another new Tree (t) from the database (db_{new}) having new transactions(considering those transactions which are entered into the database within a predefined time) and order the nodes of t lexicographically.
(3) Merge the tree t to the last updated tree T. The tree thus formed after merging t to T will have the content of the entire updated database ($db_{original}$) ∪ (db_{new}) with tree nodes ordered lexicographically (i.e. T ← T ∪ t).
(4) Continue with Step 2 when new transactions entered into (db_{new}).

Table 1. Transaction Database

Original Database		New Entries(1st Group)	
Trans Id	Transactions	Trans Id	Transactions
t1	{p, a, s, t}	t5	{ a, t, c}
t2	{ b, s, c, a, t}	t6	{ s, t, a, c }
t3	{ a, b, q, s, t}	New Entries(2nd Group)	
t4	{t, a, s}	t7	{ b, c, q, t}

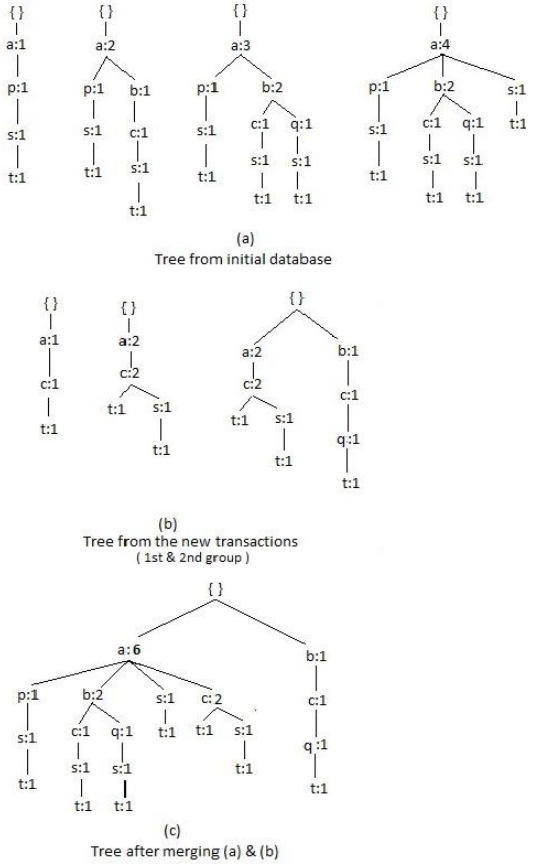

Figure 1. GMTree after adding each transaction

3.2. GMTree-GTree Algorithm

The GMTree-GTree algorithm [2] makes use of the sliding window technique[11]. A batch contains a group of transactions that is treated as a single entity. A sliding-window consists of 'k' groups of transactions, where 'k' is the window size. When a window becomes full, the earliest batch is removed and fresh batch is inserted.

The following data structures are used in this method:

(1) GMTree[5], [20] is a base tree that efficiently stores the transactions in the current window. A GMTree will have an item-table (iTable) and last-node-of-transaction-table (lTable) associated with it. Each row of iTable consists of the item id, the item's support count, and a list of nodes with this item in GMTree. Each row of lTable consists of the index of the batch, and a list of the last nodes of transactions in GMTree.
(2) GTree is a projection-tree, built from the GMTree and it is used to mine frequent patterns. A GTree also has an iTable associated with it.
(3) Comparison of new items with the original tree nodes Similar items generally form a single node in the new tree for comparison

To construct a GMTree[5], [20] the data items belonging to each transaction in the new batch are sorted canonically and then these are added to the GMTree. If each data item belonging to a transaction on a path from the root is the same as each node on the path, then the support count of the node on the path is incremented by one. Else, a new node is created, and is added to the GMTree as the child of the current node. GTree for each frequent item is constructed using the iTable of the GMTree and using the lTable, transactions of the oldest batch are discarded from the GMTree. Frequently occurring item sets of each data item is found using the GTree. From each GTree of data items, the frequent item sets starting with its root node are discovered. Sub GTrees are constructed that recursively represents its sub-problems from these. GTree is a tree that represents a group of sub trees as a single tree, where the data item in their root nodes are same as that in GMTree.

A window with 'k' batches is provided as input for this algorithm, and frequent item sets in the current sliding- window are discovered. All the

GTrees are evaluated by the algorithm, because the support count of their root item will be greater than or equal to the minimum support count. The sub-Gtrees are also traversed only when the support count of the root of a GTree is frequent. For example, only if the support count of the root of GTree A is greater than the given minimum support count, A is a frequent item set, and its sub-GTrees are constructed. A detailed example of the working of the above algorithm is available in the paper referred[2].

All frequent or infrequent data items of transactions are stored in the base-tree. The GMTree is a little different from FPTree. In GMTree the data items of a transaction are sorted in canonical order before adding them to the GMTree whereas in FP Tree the ordering is based on frequency.

Two DB scans are required in the case of FP Tree algorithm, whereas only one DB scan will be required by the GMTree. Hence for real-time applications, the GMTree-GTree algorithm works better and is more suitable than the FP-growth algorithm[20], [12], [23].

3.3. Mining Closed Frequent Item Sets, Association Rules and Implementation on Hadoop

The GMTree-GTree algorithm[26] has been modified to mine for frequently occurring closed item sets[24], [26]. Closed frequent item sets are those which are both closed and whose support is more than a minimum threshold. Consider an item set for which there does not exist any superset which has equal support count, then that item set is closed in the specifies data set.

Closed frequent item sets are used more than maximal frequent item set because when efficiency is of more importance that space, the support of the subsets is provided by them. Hence an additional pass is not required to find this information.

Knowledge discovery is also very important and is usually obtained by mining association rules [19], [11], [14]. This gives us some insight into the data and helps us to learn from it. These are basically if/then statements that support us to discover relationships between data which seems unassociated in a relational database and we can consider other data warehouses or repositories also. For example, let us consider two frequent item sets {a, c, e} with support count 2.

The dataset is illustrated in table 2 as follows:

Table 2. Sample dataset

TID	Items
100	a c d
200	b c e
300	a b c e
400	b e
500	a c e

For every non empty subset s of I, output the rule: s -> (I-s),
if supportcount(I)/supportcount(s)>= minimum confidence.
Let us suppose the minimum confidence be 60.
For R1: a, c -> e
Confidence = 2/3 = 66.66%. Rule is selected. For R2: c -> a, e
Confidence = 2/4 = 50%. Rule is rejected.

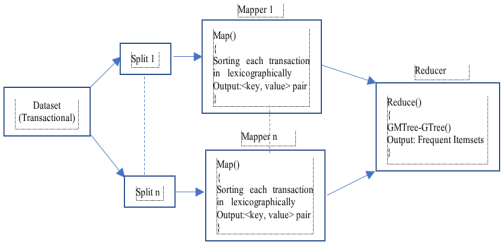

Figure 2. Hadoop GMTee-GTree flowchart

4. Results and Discussion

The GMTree-GTree algorithm was first implemented in single node using Java and was tested using the BMS- WebView-1 dataset which is a real world dataset from KDD CUP 2000 and consists of click stream data of a web store Gazelle. Then, this algorithm was implemented on the Hadoop framework after making some modifications and the time taken was compared to that of single node using Java. The dataset used for comparison was a web documents dataset "webdocs" from the FIMI repository. It is a transactional dataset that contains the main characteristics of a spidered collection of web html documents. The size of the dataset is about 1.48 GB and contains approximately 1.7 million documents. The experiments were performed on the following system: Single node using Java: Hardware: Intel core i5, 8GB RAM, CPU 2.4 GHz Software: Windows 10, Java with JDK 1.8 Hadoop implementation: Hardware: Intel core i5, 4 GB RAM, CPU 2.4 GHz Software: Ubuntu 16.10, Java with JDK 1.8, Hadoop 2.7.2 (pseudo distributed mode). The proposed system used the two different dataset

sizes. The "webdocs" is huge real-life transactional dataset which size is more bigger than KDD CUP 2000, sequences of click stream data.

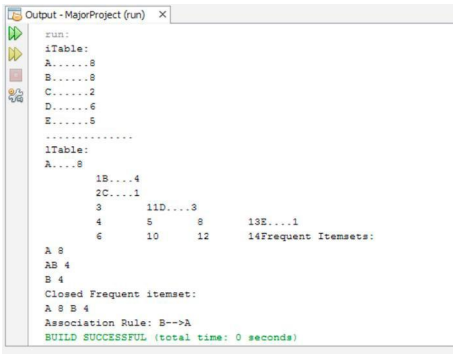

Figure 3. iTable and lTable along with complete and closed frequent item sets

Figure 3 shows the iTable and lTable of the constructed GMTree along with the complete frequent item sets, closed frequent item sets along with their support count and the association rules with 60 percent confidence and minimum support set to 4. This result is for the small dataset, as shown in Table 1. As we can see, the iTable contains the items A, B, C, etc and their frequencies. The lTable contains the item, its maximum frequency and the last nodes in which it is present, which helps us to track the transactions easily. For example, here for C, the max frequency is 1 and it is present in the nodes 3 and 11. So, from this we get the frequent item sets by eliminating the ones below the minimum support count. Also, the closed frequent item sets and the association rules are obtained by the method as discussed in section 3.2.

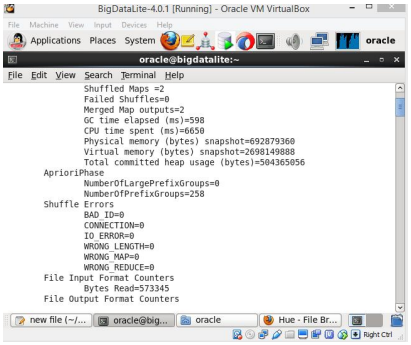

Figure 4. Hadoop implementation

Figure 4 is a screenshot when the Hadoop implementation of the GMtree-Gtree algorithm was completed successfully. The GMTree-GTree algorithm was executed in the Hadoop framework with 50 input splits, 2 maps and 2 reducer. The total execution time taken is almost 2 hours using Hadoop while it took almost 10 hours to complete execution using single node Java program.

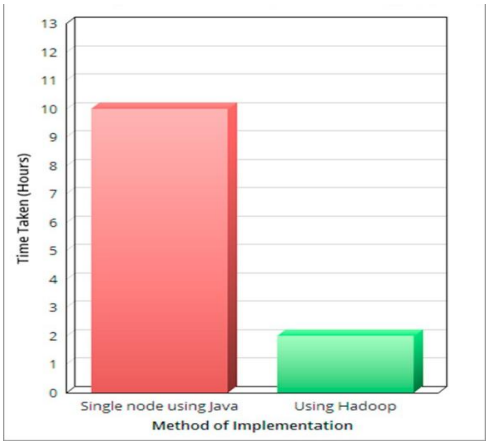

Figure 5. Time comparison of Java and Hadoop implementation

Figure 5 shows a graph that compares the execution time taken by the implementation in single node Java and Hadoop framework. It can be seen that it takes much lesser time using Hadoop as compared to simple Java execution as Hadoop is designed to handle big data efficiently and splits the given input and feeds each input split into different mappers that execute parallely. This leads to the significant time reduction in Hadoop framework as compared to a sequential execution in single node using Java.

5. Conclusion and Future Work

The Hadoop GMTree-GTree algorithm works well for mining frequently occurring patterns in real-time streaming data. As the results, the algorithm adds new transactions to the GMTree without any need for restructuring. Here, GTree is used for constructing the projection-tree in order to discover the frequently occurring item sets. So, this algorithm would be more time-efficient for mining the complete frequent item sets from dynamic, streams of data also. The GMTree-GTree algorithm execution time taken on Hadoop(pseudo distributed mode) is more less than the same proposed algorithm execute on single node Java.

As future work, Hadoop GM-Tree and GTree will be compared with another tree algorithm for data

stream mining considering the real time conditions which implemented on Hadoop.

References

[1] C. Giannella, J. Han, J. Pei, X. Yan, and P.S. Yu, Mining frequent patterns in data streams at multiple time granularities, pp. 192-209.

[2] C.H. Lee, C.R. Lin, M.S. Chen. Sliding window filtering: an efficient method for incremental mining on a time-variant database. Information Systems 2005; 30: 227-244.

[3] C. K. S. Leung and Q. I. Khan, DSTree: atree structure for the mining of frequent sets from data streams, Proc. 10th Int. Symp. Database Engineering and Applications, 2006.

[4] C. K. Leung, Q. I. Khan, Z. Li, and T. Hoque, CanTree: a canonical order tree forincremental frequent pattern mining, Knowledge and Information Systems, 2007, pp.287-311.

[5] C. K.S. Leung, Q.I. Khan, T. Hoque. CanTree: A Tree Structure for Efficient Incremental Mining of Frequent Patterns.Proceedings of the Fifth IEEE International Conference on Data Mining 2005.

[6] C. Lee, C. Lin and M. Chen, Sliding window filtering: An efficient method for incremental mining on a time-variant database. Information Systems, Vol. 30, 2005, pp. 227-244.

[7] G. Krempl, I. Zliobaite and et al., Open challenges for data stream mining research, ACM SIGKDD Explorations Newsletter, June 2014, pp. 1-10.

[8] G. Mao, Wu, X. Zhu, and G. Chen, Mining Maximal Frequent Itemsets from Data Streams, Journal of Information Science. vol. 33, no. 3, 2007, pp.251-262.

[9] H. Li, N. Zhang and Z, Chen, A Simple but Effective Maximal Frequent Itemset Mining Algorithm over Streams, Journal of Software, Vol. 7, No. 1, 2012, pp. 25-32.

[10] H. Li and S. Lee, Mining requent itemsets over data streams using efficient window sliding techniques, Expert Systems with Applications, No. 36, 2009, pp. 1466-1477.

[11] J. Chang and W. Lee, A Sliding window method for finding recently frequent itemsets over online data streams. Journal of Information Science and Engineering, Vol. 24, No. 4, 2004, pp.753-762.

[12] J. Han, J. Pei, Y. Yin, R. Mao. Mining Frequent Patterns without Candidate Generation: A Frequent-Pattern Tree Approach.Data Mining and Knowledge Discovery 2004; 8: 53–87.

[13] J. Kim, B. Hwang. Real-time stream data mining based on CanTree and GTree. Information Sciences 2016; 367 368: 512-528.

[14] L. Shen, H. Shen, L. Cheng. New algorithms for efficient mining of association rules. Inf. Sci. 1999; 118: 254–268.

[15] M. J. Zaki and C. J. Hsiao, Efficent algorithms for mining closed itemsets and their lattice structure, IEEE Trans. Knowledge and Data Engineering, vol. 17, no. 4, 2005, pp.462-478.

[16] M. J. Zaki and C.-J. Hsiao, “Charm: An efficient algorithm for closed itemset mining.” in SDM, vol. 2. SIAM, 2002, pp. 457–473.

[17] N. Jiang and Le Gruenwald, Research issues in data stream association rule mining, SIGMOD Record, Vol. 35, No. 1, Mar. 2006, pp. 14-19.

[18] P. Y. Hsu, Y. L. Chen and C. C. Ling, Algorithms for mining association rules in bag databases, Information Sciences, vol.166, 2004, pp.31-47.

[19] R. Agrawal, T. Imielinski, A.N. Swami. Mining association rules between sets of items in large databases. Proceedings of the ACM SIGMOD Conference on Management of Data 1993; 207–216.

[20] S.K.Tanbeer, C.F. Ahmed, B.S. Jeong, Y.K.Lee. Sliding window-based frequent pattern mining over data streams. Information Sciences 2009; 179: 3843-3865.

[21] T.-P. Hong, C.-W. Lin and Y.-L. Wu, An efficient FUFP-tree maintenance algorithm for record modification, International Journal of Innovative Computing, Information and Control, vol.4, no.11, pp.2875-2887, 2008.

[22] T. P. Hong, C. W. Lin and Y. L. Wu, Incremental fast updated frequent pattern trees, Expert Systems with Applications, vol.34, 2008, pp.2424-2435.

[23] T.T. Nguyen. A Compact FP-tree for Fast Frequent Pattern Retrieval. PACLIC 2013; 27.

[24] V. Kumar, S.R. Satapathy. A novel technique for mining closed frequent item sets using variable sliding window. IEEE International Advance Computing Conference (IACC) 2014; 504–510.

[25] W. Cheung and O. R. Zaiane, “Incremental mining of frequent pat- terns without candidate generation or support constraint,” in Database Engineering and Applications Symposium, 2003.

Proceedings. Seventh International. IEEE, 2003, pp. 111–116.

[26] Y. Chi, H. Wang, P.S. Yu, R.R. Muntz. Catch the moment: maintaining closed frequent item sets over a data stream sliding window. Knowl. Inf. Syst. 2006; 10 (3): 265–294.