Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 115 (2017) 266–273

Procedia Computer Science

www.elsevier.com/locate/procedia

7th International Conference on Advances in Computing & Communications, ICACC-2017, 22-24 August 2017, Cochin, India

Frequent pattern mining on stream data using Hadoop CanTree-GTree

Vanteru Kusumakumari*, Deepthi Sherigar, Roshni Chandran, Nagamma Patil

National Institute of Technology Karnataka, Surathkal 575025,India.

Abstract

The need for knowledge discovery from real-time stream data is continuously increasing nowadays and processing of transactions for mining patterns needs efficient data structures and algorithms. We propose a time-efficient Hadoop CanTree-GTree algorithm, using Apache Hadoop. This algorithm mines the complete frequent item sets (patterns) from real time transactions, by utilizing the sliding window technique. These are used to mine for closed frequent item sets and then, association rules are derived. It makes use of two data structures - CanTree and GTree. The results show that the Hadoop implementation of the algorithm performs 5 times better than in Java.

Peer-review under responsibility of the scientific committee of the 7th International Conference on Advances in Computing & Communications.

Keywords: Stream data mining; Frequent item sets; GTree; CanTree; Hadoop.

1. Introduction

Over the past few decades, numerous algorithms for discovering frequent item sets[7] have been proposed. Generally, the item set can be of two types - static or dynamic. Static data is the main focus of most algorithms. The development of sensors and their rising uses has prompted an increase of information and has turned into the most researched topic of mining data. A few necessary features of streaming data are continuity, unlimited and non-uniform distribution. They have many real-time uses like eco monitoring, share market analysis, bio-informatics,

* Corresponding author. Tel.: +91-8861117744.
E-mail address: kusuma95@yahoo.co.in.

1877-0509
Peer-review under responsibility of the scientific committee of the 7th International Conference on Advances in Computing & Communications
10.1016/j.procs.2017.09.134

health related solutions, etc require stream data mining to uncover important information. Meanwhile, data mining techniques[7] can be categorized on the basis of response time into two types - batch and real-time processing. In case of batch processing, frequently occurring item sets are discovered from static item sets, whereas in real-time methods, frequent item sets are extracted on the spot from dynamic item sets that vary with time. Specifically for data streams, a real-time method is preferred to a batch processing method.

Data streams that are produced continuously need a data analyzing technique to extract transactions from them. Mining methods on streaming data[7], [10], [17] can be split into the following types on the basis of the window concept: landmark, damped and sliding-window. Concentration is mainly on the item set that is observed from a fixed time in the past to the present time, in case of the Landmark model. In the damped approach, frequent item sets are extracted in data stream where every transaction of the data is allotted a weight that reduces with age. In the sliding-window model, the main focus is on the item sets that are collected in a certain interval of time from the present time. This sliding window moves forward when a new set of transaction(s) arrive.

The fundamentals of this algorithm are the sliding-window technique[4], [11] which moves per unit batch. There are many specifications for data stream mining, such as only one scan of the whole database, organizing and representing all the transactions in an efficient manner, and maintaining the window in proper way, by adding and deleting transactions in an effective manner. So, an algorithm with good performance is needed for efficient extraction of frequently occurring item sets in the given window. The proposed algorithm fulfills these needs. For an efficiently representing the transactions, it makes use of a base-tree that is constructed from the CanTree, which is almost the same as FPTree. The only change between the two is that when a new arrives, we need not restructure the entire tree. The data items of that transaction are just arranged canonically and then, appended to the Cantree.

To discover frequent item sets, for every frequent data item in CanTree[5], [20] we propose projection tree that is generated from the base-tree. A node of GTree represents a set of nodes that have the same data item in the base-tree (CanTree or parent GTree). Thus, the proposed algorithm called Hadoop CanTree-GTree, is very simple and efficient. The algorithm has the following properties:

- Single Database scan – Only single scan is needed to store the transactions. After the construction of tree on a set of transactions there won't be a need for restructuring because the transactions are alphabetically arranged before inserting them in the tree.
- Top-down tree traversing – FP-Growth algorithm follows bottom up traversing so the construction time is high. Hence, the performance is better in G-Tree as it uses top-down traversal.
- Usage of sliding-window – Hadoop CanTree-GTree maintains two tables for storing past transaction details. Hence, restructuring of tree is not required.
- Finding exact and complete frequent item sets[12], [13] – The proposed CanTree-GTree algorithm[2] is modified to mine the complete frequent items which would be more useful than a set a redundant frequent item sets.

2. Literature Survey

In the past few decades a large variety of algorithms have been developed to obtain frequently occurring patterns from datasets. As the volume of the data increases the computations increase hence there is a need to come up with better techniques which are both space and time efficient. In data mining, the basic task is full or partial analysis of huge volumes of data in order to extract previously not known, patterns which are useful and interesting like groups of data records, unusual records and dependencies. The first ever algorithm which was proposed for getting frequent patterns was Apriori Algorithm[1], [18]. It is one of the classical approaches to get the frequent patterns. If this algorithm is used on a big data with huge volume of transactions, this technique becomes computationally intensive. It is a two pass algorithm which means that two scans are needed.

To discover frequent patterns, the FP-Tree algorithm[3], [8], [21] is a more efficient algorithm. This algorithm uses tree data structure. Advantages of FP-Growth are that only 2 passes are required to be done. The dataset is also compressed and candidate generation is not present as in Apriori and this makes it much faster than Apriori Algorithm. Complexity is higher than that of Apriori and if the sliding window is not adopted during creation of the tree there is a possibility of running out of memory because the entire tree may not be able to reside in memory.

CPS Tree[4] is used as one of the base algorithm and the whole set of the frequently occurring patterns of the current window are provided by the FP-Growth mining technique.CPS tree method has high efficiency in terms of managing memory. The time complexity of finding the recently occurred patterns from data stream is also less. The advantage of CPS Tree[6] is its capacity of handling tilted time windows. CPS Tree can handle concept drifts and still provide good results. It can also provide maximal and minimal close patterns to the frequent pattern obtained at the end. This is supported by sliding window concept[4], [11], [12], [13].

Data Stream tree[6] also provides a feature for extracting the patterns that occur frequently, particularly for mining streams of data[2], [10], [17]. The content of the transaction is captured by the tree, while analyzing every batch of the data stream. The transaction is arranged in some canonical order which remains unaffected due to the insertion of the next batch of transaction.

Apriori algorithm[1], [18] is the one that is the simplest and used popularly. But, repeated database scans were a problem that was overcome by the FP-Growth algorithm. CPS and DS Trees provide some basics regarding use of the primitive algorithms in creating complex trees. This variety of algorithms use different methods to discover the frequent patterns hence there is a need to think of an algorithm which would have all the advantages of the above algorithm and be time efficient.

3. Proposed Methodology

3.1. CanTree-GTree Algorithm

The CanTree-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. CanTree[5], [20] is a base tree that efficiently stores the transactions in the current window. A CanTree 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 CanTree. Each row of lTable consists of the index of the batch, and a list of the last nodes of transactions in CanTree.

2. GTree is a projection-tree, built from the CanTree and it is used to mine frequent patterns. A GTree also has an iTable associated with it.

To construct a CanTree[5], [20] the data items belonging to each transaction in the new batch are sorted canonically and then these are added to the CanTree. 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 CanTree as the child of the current node. GTree for each frequent item is constructed using the iTable of the CanTree and using the lTable, transactions of the oldest batch are discarded from the CanTree.

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 CanTree.

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 CanTree is a little different from FPTree. In CanTree the data items of a transaction are sorted in canonical order before adding them to the CanTree 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 CanTree. Hence for real-time applications, the CanTree-GTree algorithm works better and is more suitable than the FP-growth algorithm[3], [8], [21].

3.2. Mining closed frequent item sets, association rules and implementation on hadoop

The CanTree-GTree algorithm[2] has been modified to mine for frequently occurring closed item sets[12], [13]. 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[9], [16], [18]. 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 {1, 3, 5} with support count 2. The dataset is illustrated in table 1 as follows:

Table 1: Sample dataset

TID	Items
100	1 3 4
200	2 3 5
300	1 2 3 5
400	2 5
500	1 3 5

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: 1, 3 -> 5

Confidence = 2/3 = 66.66%. Rule is selected.

For R2: 3 -> 1, 5

Confidence = 2/4 = 50%. Rule is rejected.

The CanTree-GTree algorithm is implemented on Hadoop framework[14], [15], [22] in order to achieve better performance. Hadoop is a popular open source framework and the foundation/core is built on Java programming. One of its main functions is processing and storage of huge volumes of data in a distributed computing environment. Apache Hadoop mainly comprises of two parts: storage, known as Hadoop Distributed File System (HDFS), and a processing part which is a MapReduce programming model[15], [22]. Initially, files are divided into huge blocks and are distributed across the various nodes in the cluster. The packaged code is then transferred into nodes to process the data in parallel. This is what causes the time required for execution to significantly reduce. The CanTree-GTree algorithm is slightly modified in order to execute in the MapReduce environment. The dataset is split using input split. After doing this, the mapper gives each transaction in the (key, value) pair format and this is feeded to the reducer. The reducer constructs the CanTree and GTree and mines frequent patterns from it.
Each mapper will be provided with an input split. The number of mappers depends on how many input splits are done. All the mappers will process the data in parallel. The map function basically reads each transaction and the data items are sorted in every transaction canonically. The sorted transactions forms the output of the mapper as a <key,value> pair. The output of the mapper goes through the shuffle, sort and merge phases, where similar keys are sorted and their values merged. The reducer constructs the CanTree and GTree from the sorted transactions and the CanTree-GTree algorithm is needed for mining the frequently occurring frequent patterns.

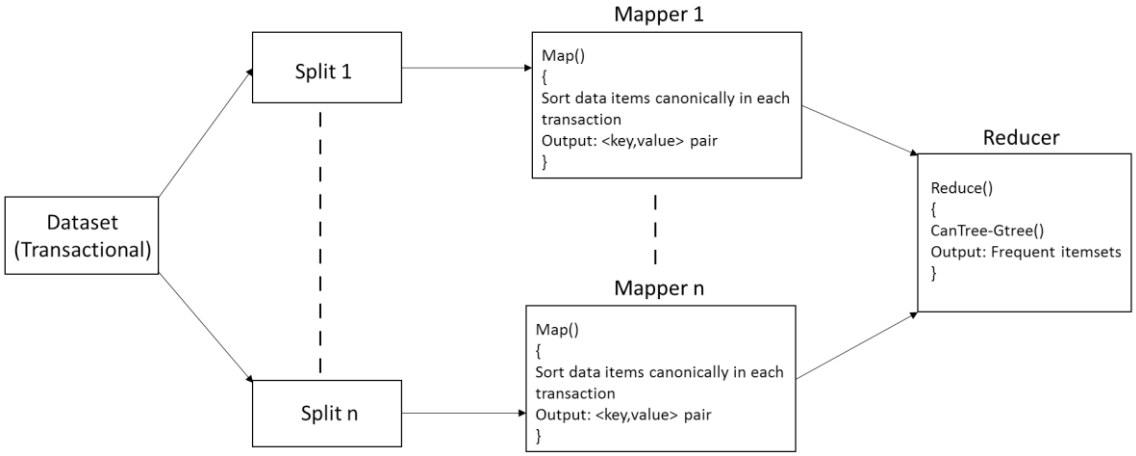

Fig 1: Hadoop Cantree-Gtree flowchart

4. Results and Discussion

The CanTree-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 webstore 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 17 lakh transactions. 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 2.7.0 (pseudo distributed mode).

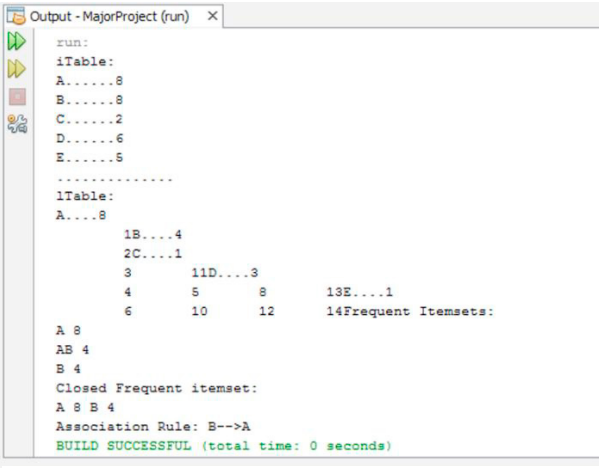

Fig 2: iTable and lTable along with complete and closed frequent item sets

Figure 2 shows the iTable and lTable of the constructed CanTree 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.

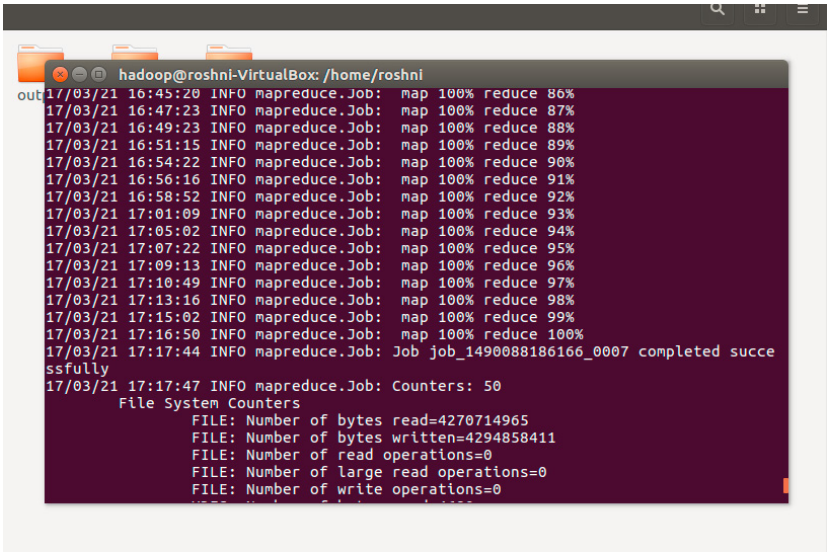

Fig 3: Hadoop implementation

Figure 3 is a screenshot when the hadoop implementation of the Cantree-Gtree algorithm was completed successfully. The CanTree-GTree algorithm was executed in the Hadoop framework with 45 input splits and 1 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.

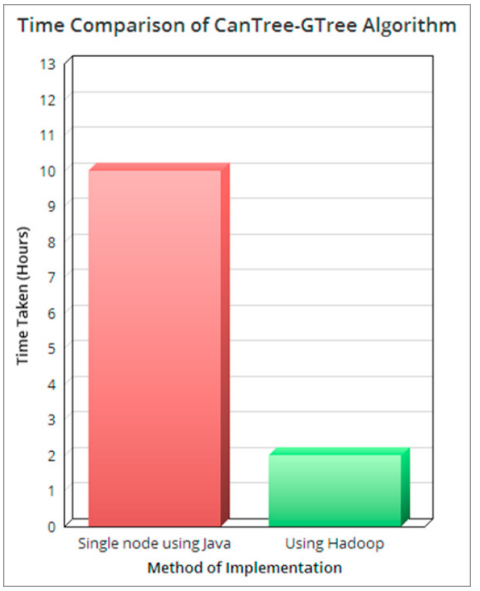

Fig 4: Time comparison of Java and Hadoop implementatin

Figure 4 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

From the results obtained, we can observe that the Hadoop CanTree-GTree algorithm works well for mining frequently occurring patterns in real-time streaming data. As the window slides, the algorithm adds new transactions to the CanTree 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. Using the same proposed algorithm in a MapReduce framework significantly lowers the execution time taken.

As future work, other tree algorithms for data stream mining considering the real-time conditions will be implemented in Hadoop or any other similar frameworks like Apache Spark, Storm etc.

6. References

[1] Rakesh Agrawal, Ramakrishnan Srikant. Fast Algorithms for Mining Association Rules. Proceedings of the 20th VLDB Conference Santiago 1994.
[2] Jaein Kim, Buhyun Hwang. Real-time stream data mining based on CanTree and GTree. Information Sciences 2016; **367 368**: 512-528.
[3] Syed Khairuzzaman Tanbeer, Chowdhury Farhan Ahmed, Byeong-Soo Jeong, Young-Koo Lee. Sliding window-based frequent pattern mining over data streams. Information Sciences 2009; **179**: 3843-3865.
[4] Chang-Hung Lee, Cheng-Ru Lin, Ming-Syan Chen. Sliding window filtering: an efficient method for incremental mining on a time-variant database. Information Systems 2005; **30**: 227-244.
[5] Carson Kai-Sang Leung, Quamrul I. Khan, Tariqul Hoque. CanTree: A Tree Structure for Efficient Incremental Mining of Frequent Patterns. Proceedings of the Fifth IEEE International Conference on Data Mining 2005.

[6] Carson Kai-Sang Leung, Quamrul I. Khan. DSTree: A Tree Structure for the Mining of Frequent Sets from Data Streams. Proceedings of the Sixth International Conference on Data Mining 2006.
[7] Jiawei Han, Micheline Kamber, Jian Pei. Data Mining Concepts and Techniques. Morgan Kauffman 2012.
[8] Jiawei Han, Jian Pei, Yiwen Yin, Runying Mao. Mining Frequent Patterns without Candidate Generation: A Frequent-Pattern Tree Approach. Data Mining and Knowledge Discovery 2004; **8**: 53–87.
[9] 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.
[10] T. Calders, N. dexters, J.J.M. Gillis, B. Geothals. Mining frequent item sets in a stream. Information Systems 2014; **39**: 233–255.
[11] J. Chang, W. Lee. A Sliding window method for finding recently frequent item sets over online data streams. J. Inf. Sci. Eng. 2004; **24** (**4**): 753–762.
[12] 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.
[13] 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.
[14] A. Cuzzocrea, C.K. Leung, R.K. Mackinnon. Mining constrained frequent item sets from distributed uncertain data. Future Gener. Comput. Syst. 2014; **37**: 117–126.
[15] F. Fumarola, D. Malerba. A parallel algorithm for approximate frequent item set mining using Mapreduce. Int. Conf. High Perform. Comput. Simul. (HPCS) 2014.
[16] N. Jiang, Le Gruenwald. Research issues in data stream association rule mining. SIGMOD 2006; **35**: 14–19.
[17] J.H. Chang, W.S. Lee. estWin: Online data stream mining of recent frequent item sets by sliding window method. J. Inf. Sci. 2005; **31** (**2**): 76–90.
[18] L. Shen, H. Shen, L. Cheng. New algorithms for efficient mining of association rules. Inf. Sci. 1999; **118**: 254–268.
[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] C.K. Leung, Q.I. Khan, Z. Li, T. Hoque. CanTree: a canonical order tree for incremental frequent pattern mining. Knowl. Inf. Syst. 2007; 287–311.
[21] Tri Thanh Nguyen. A Compact FP-tree for Fast Frequent Pattern Retrieval. PACLIC 2013; 27.
[22] Sanket Thakare, Sheetal Rathi, R.R. Sedamka. An Improved PrePost Algorithm for Frequent Pattern Mining with Hadoop on Cloud. 7th International Conference on Communication, Computing and Virtualization 2016; **79**: 207–214.