Project Report On

A PARALLELIZED INCREMENTAL FREQUENT PATTERN MINING ALGORITHM FOR UNCERTAIN DATA

Submitted by

Thejaswini D M (171IT243)

VI SEM B.Tech (IT)

Under the guidance of

Jaidhar C.D.
Dept of IT, NITK Surathkal

in partial fulfillment for the award of the degree

of

Bachelor of Technology

in

Information Technology

at

Department of Information Technology

National Institute of Technology Karnataka, Surathkal.

June 2020

Department of Information Technology

National Institute of Technology Karnataka, Surathkal

Mid Semester Evaluation (February 2020)

Course code : IT 399

Course Title: Minor Project

Title of the Project: A parallelized incremental frequent pattern mining algorithm for uncertain data

Details of Project Group

Name of the Student	*Register No.*	*Signature with Date*
1. Thejaswini D M	171IT243	

Name of Project Guide: Jaidhar C.D.

Signature of the Project Guide:

Place: NITK, Surathkal, Karnataka

Date: 15/06/20

ABSTRACT

Association rule analysis is an important topic in data mining. Most studies find frequent itemsets from traditional transaction databases, in which the contents of each transaction are definitely known and price. However, there are many situations in which ones are uncertain. This calls for mining uncertain data. Moreover, floods of uncertain data can be produced in many other situations. This leads to stream mining of uncertain data. Basket analysis is one of the most well-known applications. Store or retailer can get better sales through the analysis of goods combination. For example, placing beer and diapers at the same place can bring greater sales for the store. However, due to the rapid increase in the amount of data in this big data era, how to mine frequent patterns from big data has become an important issue. Many approaches were proposed to solve the incremental problem of certain data, but these approaches did not address uncertain data.The CUF-Growth algorithm preventing branches improves the performance of the traditional UF-Growth. Implementation and parallelization of an incremental association algorithm based on CUF-Growth to solve the problem of incremental updating of uncertain frequent items is done. This method retains the advantages of the original CUF-Growth, and significantly reduces the complexity of adding new transactions.

Table of Contents

LIST OF FIGURES

Chapter 1: INTRODUCTION

1.1. Motivation

Frequent itemset mining plays an important role in the mining of various patterns and is in demand in many real life. Most studies find frequent itemsets from traditional transaction databases, in which the contents of each transaction are definitely known and price. However, there are many situations in which ones are uncertain. This calls for mining uncertain data. Moreover, floods of uncertain data can be produced in many other situations. This leads to stream mining of uncertain data.

In recent years, uncertain data has become ubiquitous because of new technologies for collecting data which can only measure and collect the data in an imprecise way. Furthermore, many technologies such as privacy-preserving data mining create data which is inherently uncertain in nature. As a result there is a need for tools and techniques for mining and managing uncertain data. Due to which Data mining of uncertain data has become an active area of research recently. In real world scenario, new transactions are added to the existing database periodically which increases the need for an algorithm which can modify the tree data structure to give results faster instead of reconstructing the data structure again whenever a new transaction is added to the old huge uncertain data.

Chapter 2: LITERATURE REVIEW

2.1 Existing Certain Data Algorithms

2.1.1 Apriori Algorithm

In the database, it is assumed that if the set of transaction items of the Transaction Identifier (TID) is larger than the original threshold, we will refer to the collection of items as a frequent item set. The threshold value is defined by users.

Agrawal et al. proposed Apriori algorithm[1], which candidate sets are generated and are compared with the data in the database. The followings are the steps of the Apriori algorithm:

- The first step: Scan the entire transaction database and calculate the support of all generated candidate itemsets. Then remove the itemsets less than threshold and obtain the n-candidate frequent itemset.
- The second step: Combine the previous n-candidate frequent itemset with each other and generate n+1-candidate frequent itemset.
- Repeating first step and second step until no new n-candidate frequent itemset can be generated.

The biggest drawback of the Apriori algorithm is that it is time consuming to recursively scan the database. Assuming that the amount of data becomes larger, it takes a lot of time to read the database repeatedly, and each stage has a large number of frequent itemsets, resulting in bad efficiency.

2.1.2 Fp-Growth Algorithm

Since Apriori algorithm needs to rescan the database in each stage. It causes lots of I/O time consumption. Many scholars made improvements to the Apriori algorithm , but results are still not very effective.

FP-growth algorithm[2] used tree structure to improve efficiency. It is not like the Apriori algorithm in which a large number of candidate itemsets are generated. The FP-Growth algorithm can be divided into two parts. The first part is to create the tree, and the second part

is to explore. In the exploration of frequent itemsets, the FP-Tree structure is used to recursively find out all the frequent itemsets.

- Build FP-tree: The first step: Scan the database, through the item of occurrence to explore more than or less than of the threshold. Then built item occurrence counts in the Head Table. The second step: Scan the database again and build FP-Tree based on the Head Table.
- Mining FP-tree: The first step: For each branch node of FP-Tree, create a Conditional Pattern Base. The second step: Base on Conditional Pattern Base create Conditional FP-Tree. The third step: The leaf node to root Bottom up mining by recursive. The fourth step: Conditional FP-Tree contains a complete association path that lists all patterns.

2.2 Existing Uncertain data Algorithms

2.2.1 UF-Growth Algorithm

In uncertain data, each transaction contains different items and these items come with probabilities. However, the same probability can appear in different items. Leung et al. propose UF-Growth method [3] in 2008. UF-Growth method addresses the uncertain data problem. This method is similar to the FP-Growth build tree process and can be applied to uncertain data. However, UF-growth uses tree structure to store the possibilities of different items. In UF-tree, items with the same probability in different transactions can be merged into a single node. In uncertain data, the item probabilities usually are different, so items with different probabilities become different nodes.

2.2.2 Cuf-Growth Algorithm

In the past, UF-Growth algorithms solve the tree growth of uncertain data, but this method produces different branches when encountering different probabilities. This will result in excess resource consumption. When large amounts of data are available, the UF-Growth constructed tree structure will become too large and the construction process is inefficient.

Leung et al. propose CUF-Growth in 2012. This method can effectively reduce the number of nodes in different probability problems of uncertain data. Not only can it improve the efficiency of tree structure construction, but also it has increased the speed in the mining step.

Therefore, compared to UF-Growth, CUF-Growth performs better the traditional UF-Growth. This method adopts the concept of transaction cap. The transaction cap is the highest probability of different items in the transaction record. The tree structure is constructed based on the transaction caps. The probabilities of the descendent nodes are calculated from the cumulative transaction cap probabilities when the tree structure is constructed.

2.2.3 IFPM-BS Algorithm

Dong et al. proposed IFPM-BS(Incremental Algorithm For Frequent Pattern Mining Based On Bit-Sequence) algorithm in 2011. The algorithm uses the concept of pre-frequent items in the mining association. The total transaction is divided into the upper reaches of the Support Value of 50%, while the lower bound of 30%, assuming there are total of 10 transactions, the counts of threshold ≥ 5 means frequent items, if the counts of threshold <3 means infrequent item, between 3-5counts it should be called the pre-frequent items set. IFPM-BS algorithms that the bit sequence table indicates that the bit index value in the transaction entry to check if the transaction had been purchased.

2.2.4 RUFP Algorithm

Mundra et al. propose Rapid Update In Frequent Pattern(RUFP) algorithm[4] in 2013. This method is based on Apriori algorithm for improvement. The RUFP algorithm is divided into two parts, the first part is the traditional data format. Each transaction data records the counts of each field and gets the threshold of each item, then get the first sort of frequent items, frequent items with F (Frequent) that infrequent items are expressed in IF (Infrequent). The second stage uses the intersection to obtain frequent itemsets. In contrast, the traditional Apriori algorithm must scan the entire database before it can get the transaction, so the RUFP algorithm effectively reduces the time of each frequent project set to scan the database to achieve a more efficient algorithm.

2.3 Outcomes of Literature Review

- Drawback of Apriori Algorithm is that it takes more time
- Drawback of the Fp-tree algorithm is it requires huge memory space for big data.
- In UF-Growth algorithm solve the tree growth of uncertain data, but this method produces different branches when encountering different probabilities.
- In the CUF-Growth algorithm, IFPM-BS algorithm, RUFP algorithm the processing of increasing datasets cannot be done. These algorithms need to be executed again from the beginning which consumes a lot of time.

2.4 Problem Statement

Previous research on the association mining algorithm actually consumes a lot of resources in the database scan process. When new data is added to the original data, the tree is reconstructed once again.

When dealing with the big data, data are constantly added and modified, if the tree is constantly reconstructed in such a traditional manner, it will decrease the efficiency a lot, so there is a need for a method to retain the tree structure in uncertain data, to maintain the flexibility of the original data, and should not require to rebuild the tree again.

2.5 Research Objectives

An algorithm which processes the uncertain data which keeps getting expanded with time without having to reconstruct the whole tree again is required to deal with big data, data that is constantly expanding without having to reconstruct the whole tree again .

Along with this, the initial step of counting the total probabilities of all the items should be done faster as that step is the bottle-neck of the whole process, this can be done by using parallelization technique using pools and maps to achieve high efficiency, better runtime, and also achieve a good speedup.

Chapter 3. METHODOLOGY AND FRAMEWORK

3.1 System Architecture

The uncertain dataset used here is randomly generated.Initially uncertain data is loaded, then the transaction cap of all transactions are calculated, along with accumulated frequency which is done parallely using GPU. This parallelization is done using Multiprocessing library in Python which imports maps to count and store the total probability of all the items.

Tree is constructed based on the frequent item arrangement in descending order. A frequent pattern is mined at this stage.

Updation of tree data when a new set of transactions are added to the existing uncertain data is done along with the adjustment of the tree nodes order by splitting and reordering methods. Finally the new frequent pattern is mined and displayed .

3.2 Method Description

The proposed method can be divided into three steps, (1) database scan, (2) adding new transactions to the accumulating table and conducting probability updates, and (3) add new transactions into the original tree structure. The following describes the three steps in detail:

- *The first step*: Scan the new transaction items in the data, and then add the items into accumulating table parallely, and then check the items sorted in the created table.
- *The second step:* This step can be divided into three parts, splitting, sorting and merging. New items are sorted according to the accumulating table. Traverse the parent node, and perform the splitting process without affecting the probabilities of other child nodes.
- *Adjusting:* New item order can be obtained from the previous step. The nodes are exchanged with each other here.
- *Merging:* This step merges the items with the same level of uncertainties to a single node.
- *Tree Updating :* This step updates the tree with the new transactions added.

3.3 Detailed Design Methodologies

- *First Step:*

  This step involves loading the uncertain data into the memory, constructing a node with attributes like frequency, value, total probability for each item in all the transactions. Counting the probabilities and frequency of all the different itemsets parallely using multithreading library of python and sorting them in descending order, accordingly sort the items in each of the transactions as well and construct the tree.Then update the accumulation table for the newly added transactions to the previously existing uncertain data.

- *Second Step:*

  This involves arranging the new accumulation table in descending order of the new probabilities. Based on the new arrangement, previously constructed tree is modified using splitting based on the new sorted itemsets as shown in Figure 1.

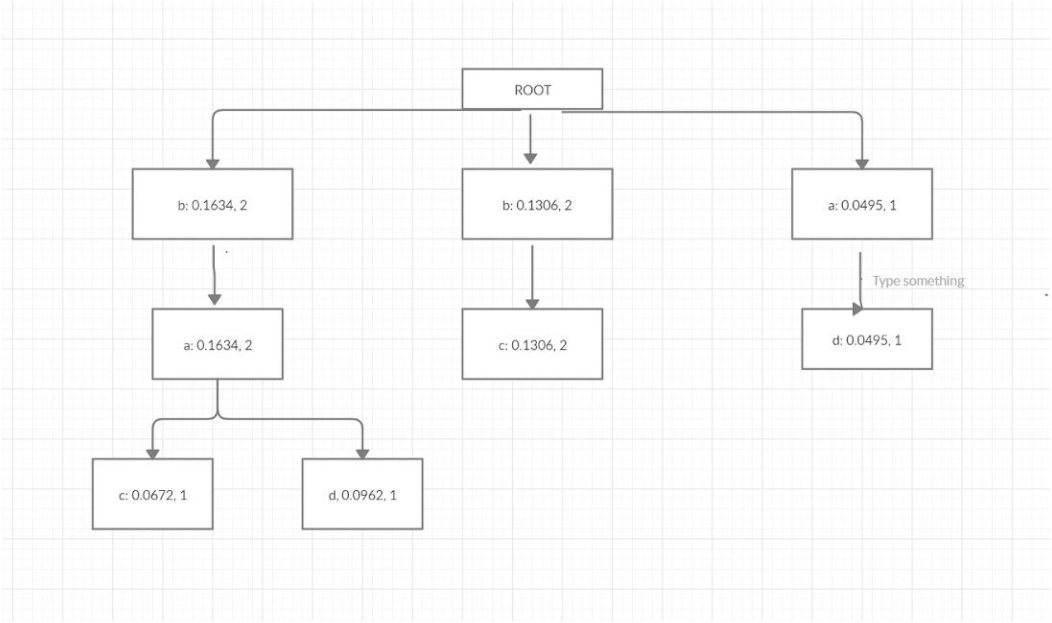

Figure 1. Splitting

- *Adjusting:*

Here the nodes are swapped based on the new accumulation table order as shown in figure 2.

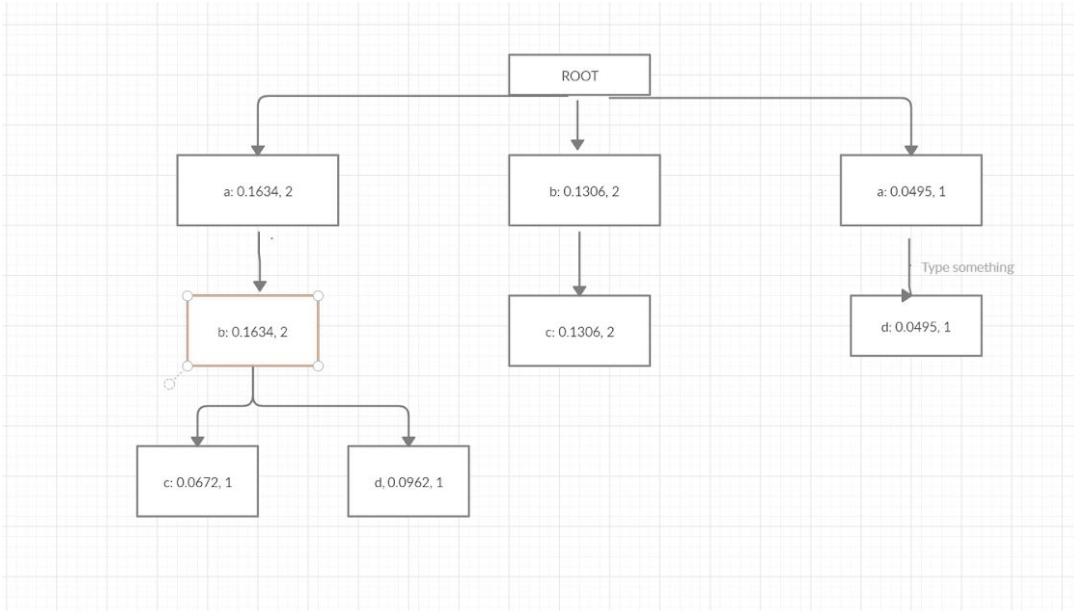

Figure 2. Adjusting

- *Merging:* This step merges the items with the same level of uncertainties to a single node in Figure 3.

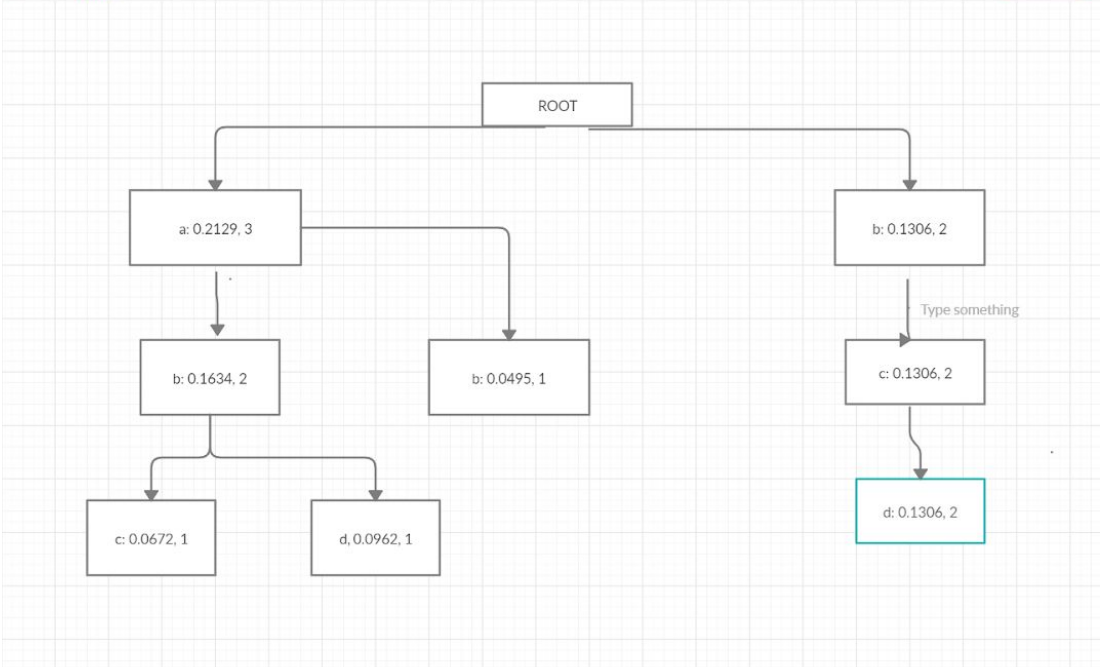

Figure 3. Merging

- *Tree Updating*

This step will add items of the new transactions into the tree structure to achieve rapid incremental data update based on CUF-Growth in uncertain data. The items are added sequentially until the new data table is empty, as in Figure 4.

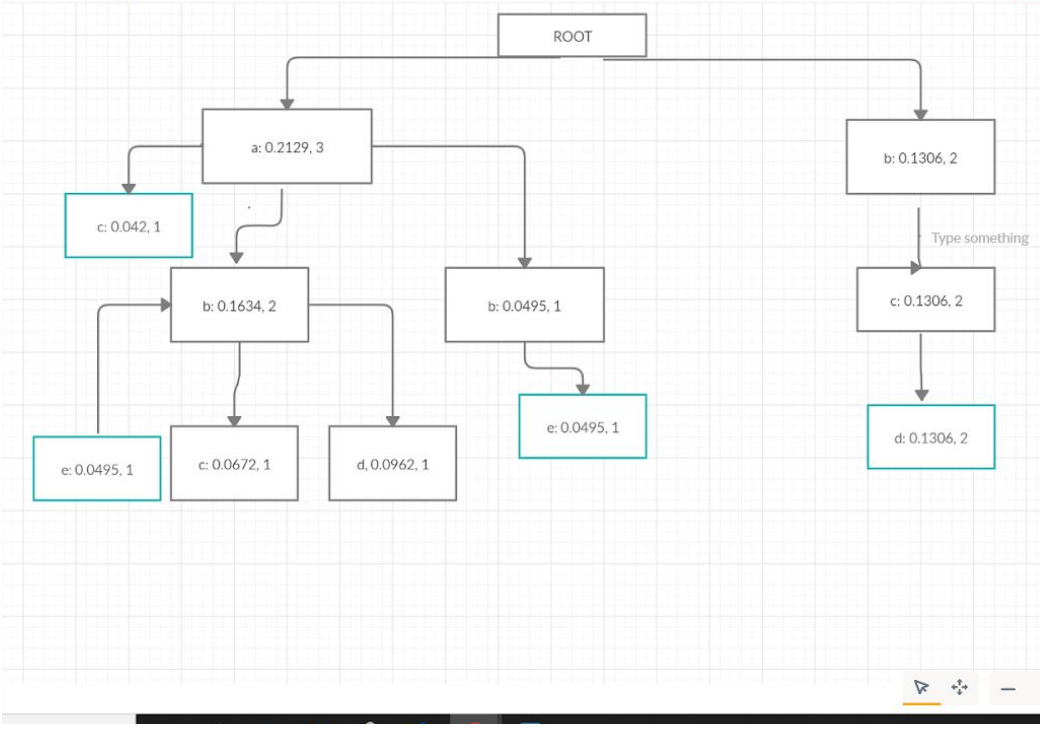

Figure 4. Tree Updating

Chapter 4: WORK DONE

4.1 Dataset:

Random Generation of dataset is done here, as of now certain dataset is being used for initial development stages, but later the uncertain dataset will be generated.

4.2 Defining the node structure and attributes:

The node of each tree containing attributes like value, parent node, children nodes, frequency is needed to be defined.

4.3 Creating the initial tree data structure:

Here the counting of all the frequency of itemsets and sorting them in the descending order, arranging each of the items in all the transactions based on the accumulation table is done.

Creation of the tree using all the transactions in the sorted order according to the accumulation table created is completed.

4.4 Loading the new transactions:

The new transactions in the existing data are loaded into the memory and the new frequency all the existing items are updated along with the new items added to the list through these new transactions.

4.5 Updating the accumulation table:

The existing accumulation table based on the previous data is updated with the information of new transactions obtained.

4.6 Splitting:

Splitting the above constructed tree based on the new accumulated table

4.7 Adjusting:

Adjusting all the paths based on the new frequency order of the accumulation table

4.8 Merging:

Merging the constructed tree based on the similar nodes

4.9 Parallelizing the counting part:

Parallelization of the very first module which involves counting the frequency of each item.

4.10 Analyzing the parallelized code:

Constructed graphs on speedup, runtime of the whole code and parallelized module.

Chapter 5: Result and Analysis:

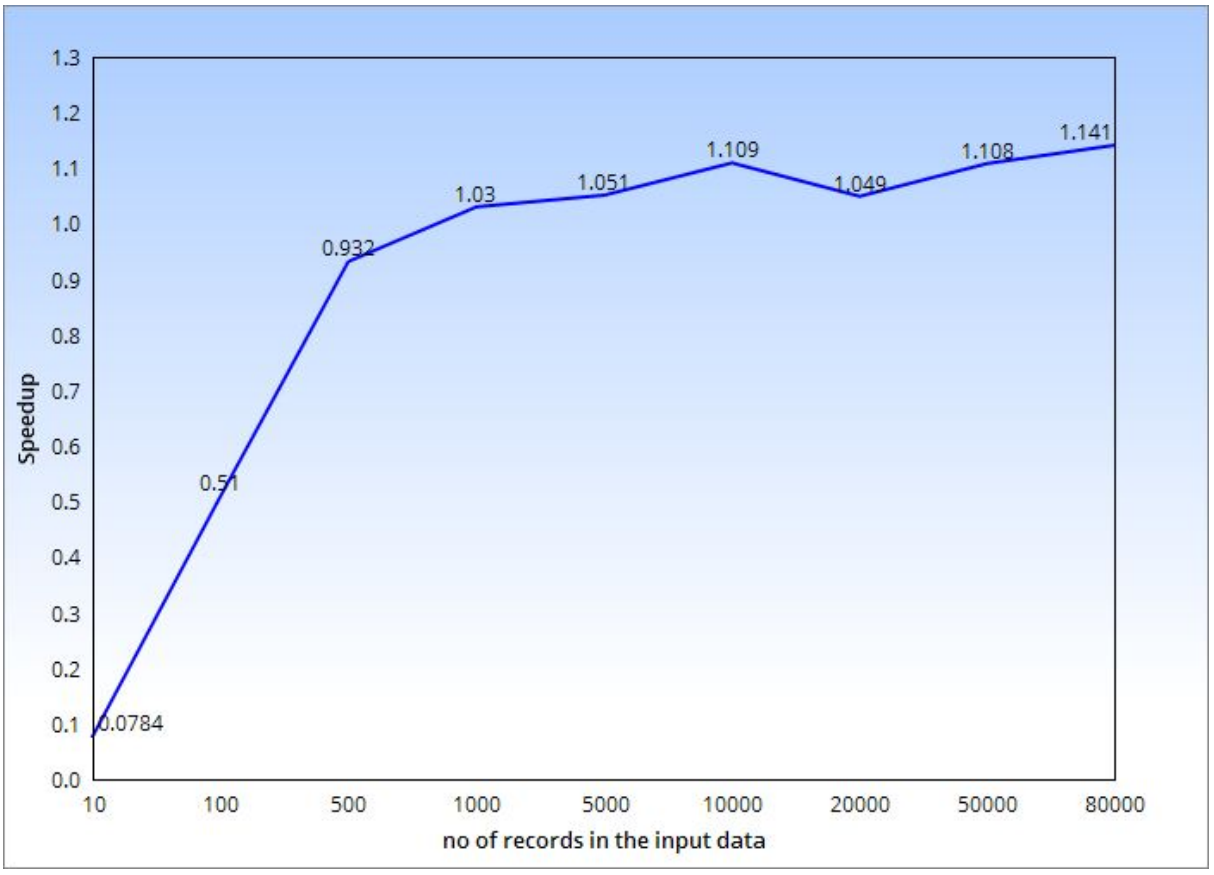

Figure 5: Speedup analysis

As shown in the graph above, the speedup is less than 1 in the initial stages where the input data size is less which in turn says that parallel code is taking more time than the serialized code.

This is due to the extra time consumed by the process to feed the input, fetch the data from different threads and then combine all the collected data in such a way that the integrity of the data received is maintained.

When the input data becomes huge with increasing data, the speedup increases as the time taken by the process to feed the input, fetch the data from different threads and then combine all the collected data in such a way that the integrity of the data received is maintained can be ignored as the work is divided into chunks and the chunks are executed parallely as the

processing chuck input does not depend on the output of the previous chunk the integrity and correctness of the output is intact.

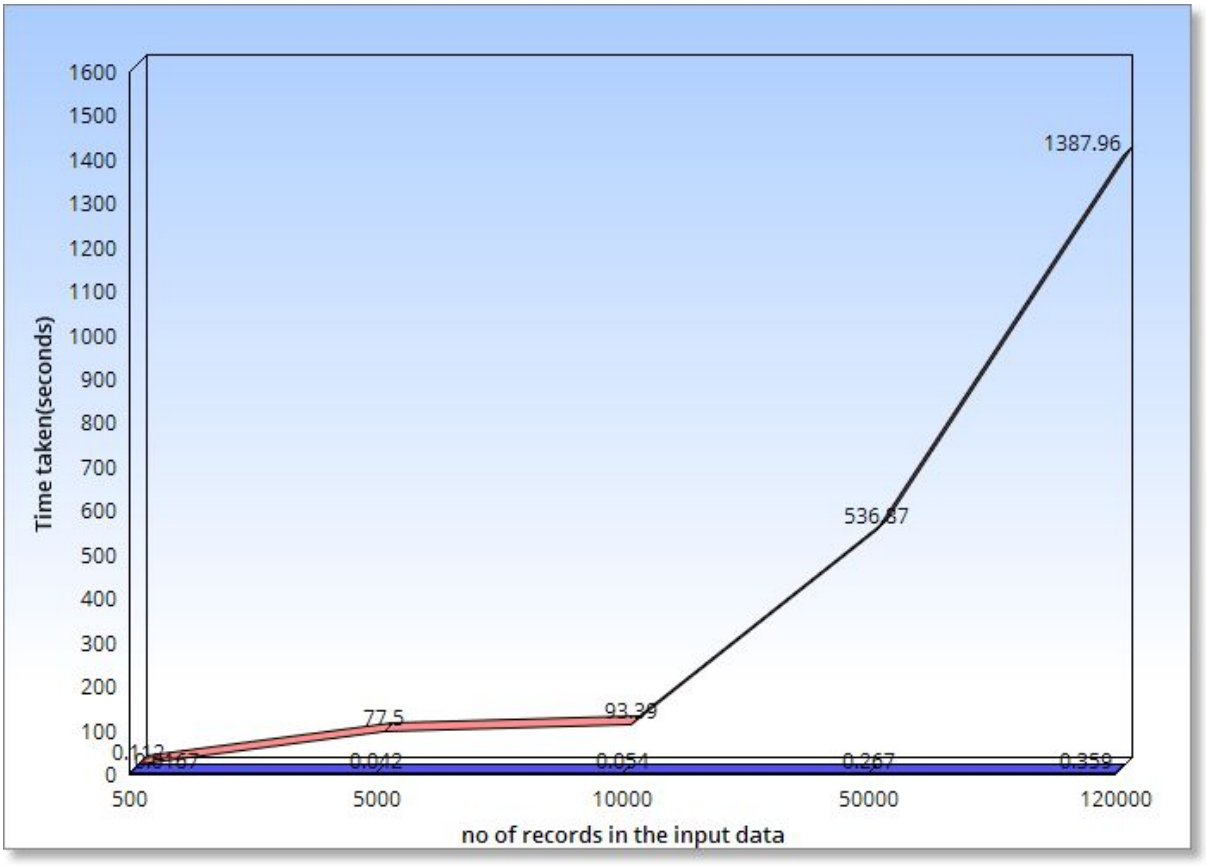

Figure 6: Runtime Analysis of parallelized module

The Runtime taken by the parallelized module alone is in fraction of a second even when the number of transactions in the input data keeps increasing.

When compared to the serial execution of the parallelized module alone, it takes a lot of time.

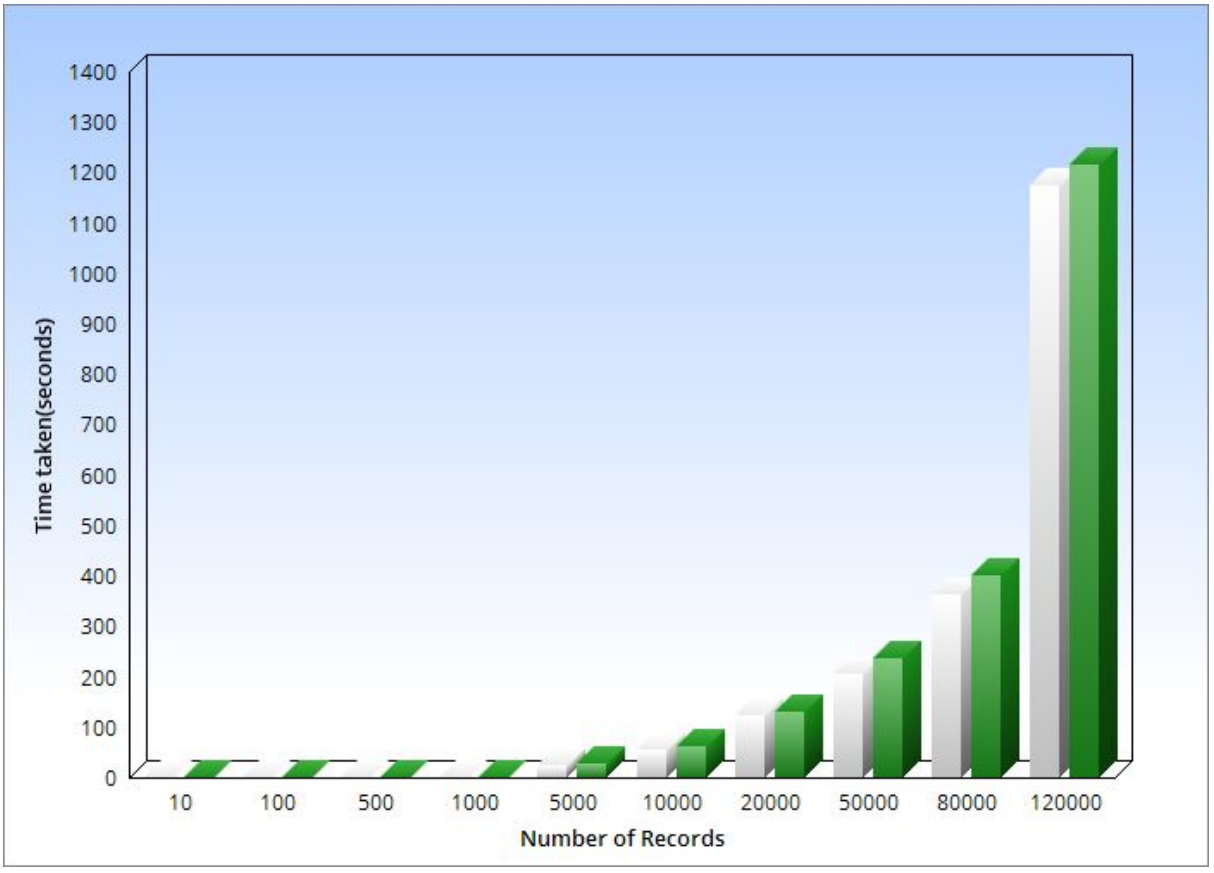

Figure 7: Time taken analysis of the whole program

The above figure shows how the runtime of both serial and parallel code varies with increasing transactions in the input file.

The maximum achieved difference in run time includes 40 seconds i.e, parallelized execution of the code takes 40 seconds less to do the same assigned task as that of serialized code.

From the above graphs, it can be concluded that the parallelized code shows 2%improvisation when compared to serialized code for big data.

Along with this when the big data gets expanded with time, the necessity of reconstructing the whole tree is removed by the methodology described above, hence providing a way to manipulate and adjust the tree to fit the new records into the same tree.

REFERENCES

[1]R. Agrawal, R. Srikant, "Fast algorithms for mining association rules", *Proc. 20th int. conf. very large data bases VLDB*, pp. 487-499, 1994.

[2]J. Han, J. Pei, Y. Yin, R. Mao, "Mining frequent patterns without candidate generation: A frequent-pattern tree approach", *Data mining and knowledge discovery*, vol. 8, pp. 53-87, 2004.

[3] C. Leung, M. Mateo, D. Brajczuk, "A tree-based approach for frequent pattern mining from uncertain data", *Advances in Knowledge Discovery and Data Mining*, pp. 653-661, 2008

[4]A. Mundra, P. Tomar, D. Kulhare, "Rapid Update in Frequent Pattern form Large Dynamic Database to Increase Scalability", *International Journal of Soft Computing and Engineering (IJSCE) ISSN*, pp. 2231-2307.

[5]https://www.semanticscholar.org/paper/Frequent-pattern-mining-with-uncertain-data-Aggarwal-Li/7d907d8ffae265bec4c200e0dfeae9b7258c4f36

[6]https://link.springer.com/chapter/10.1007/978-3-319-56991-8_73

APPENDIX