Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 45 (2015) 111 – 117

Procedia Computer Science

International Conference on Advanced Computing Technologies and Applications (ICACTA-2015)

Parallel text mining in multicore systems using FP-Tree algorithm

Krishna Gadia[1], Kiran Bhowmick[2]

[1]Student at DJ Sanghvi college of Engineering, U-15 Bhakti Vedant Swamy Marg, Mumbai – 400056, India
[2] Faculty at DJ Sanghvi college of Engineering, U-15 Bhakti Vedant Swamy Marg, Mumbai – 400056, India

Abstract

Frequent keyword mining (FKM) is a useful tool for discovering frequently occurring keywords in data. Many algorithms have been developed to speed up mining performance on single core systems. Unfortunately, when the dataset size is huge, both the memory use and computational cost can still be extremely expensive. In our paper, we try to parallelize the FP-Growth algorithm on multicore machines. We partition the huge database, into the number of cores, and utilize the combined strength of all the cores, to achieve maximum performance. We propose to use the generated FP Tree and its rules for the Trend analysis of news data.

Peer-review under responsibility of scientific committee of International Conference on Advanced Computing Technologies and Applications (ICACTA-2015).

Keywords: Parallel FP-Growth; Data Mining; Frequent Keywords Mining; Trend analysis; Multicore Mining

1. INTRODUCTION

In this paper, we try to use all the computational power available to us in this modern age. Multi-core machines can be dated back to 2001, like the IBM's POWER 4, but many of the tasks are yet not designed for multi-core machines. Although recent work deals with working with multiple machines, it does not consider that each machine in the cluster can be composed of multiple cores.

Existing frequent data mining algorithms such as Apriori and FP Growth which are ideally designed for single core machines [1][3], can be resource intensive when working with huge databases. We can use the complete potential of multicore machines to minimize the computational cost on each core. Between Apriori and FP-Growth, FP Growth runs faster, hence it is logical to work with it and make it compatible for multi-core machines. Recent work on the Algorithms try to achieve speedup on single core machines using time-slicing and threads.

Instead, we propose a solution to fork/join the task, and distribute it over multiple cores so as to achieve maximum parallel utilization. Since all tasks are usually not equally computationally heavy, work-stealing algorithm can be used, which essentially does not need the tasks to be pre-assigned to a given core. Also if the task is pre-assigned, the

Peer-review under responsibility of scientific committee of International Conference on Advanced Computing Technologies and Applications (ICACTA-2015).
doi:10.1016/j.procs.2015.03.100

framework allows the other core to steal the undone given tasks from a busy core. This ensures that none of the cores are ideal, and hence gives us maximum utilization of the resources. Because it splits into a tree structure you do only log $_2$(n) merges as opposed to n merges with linear thread splitting. Hence the join step becomes less time intensive, giving us overall faster operation.

For trend analysis [4] in text, the text can be fed into the system directly, and the system recognizes the most frequent keywords, and all the relations to them. Here the fork/join comes in handy, as it does not require all the text to be of comparable length. When converting the sentence to words, the system will also check for non-important data, such as prepositions, conjunctions, etc., so as to remove them and only identify the keywords.

Our proposed algorithm can be used both, as a standalone algorithm, or in combination with parallel processing in multiple machine system [2]. When used in the later scenario, it provides additional benefit to the overall system.

1.1 Related Work

Some previous efforts parallelized the FP-Growth algorithm across multiple threads [6] but with shared memory. However, to our problem of processing huge databases, these approaches do not address the bottleneck of huge computation on a single core. People have also worked on memory method optimization [7], but in today's world, memory is easily available but there is a lot of constraint on time.

Instead, we propose a solution to fork/join [5] the task, and distribute it over multiple cores so as to achieve maximum parallel utilization. Since all tasks are usually not equally computationally heavy, work-stealing algorithm can be used, which essentially does not need the tasks to be pre-assigned to a given core. Also if the task is pre-assigned, the framework allows the other core to steal the undone given tasks from a busy core. This ensures that none of the cores are ideal, and hence gives us maximum utilization of the resources. Previous methods spit the task in linear fashion, which takes lot of time to join the results. Our approach splits into a tree structure you do only log $_2$(n) merges as opposed to n merges with linear thread splitting. Hence the join step becomes less time intensive, giving us overall faster operation.

2. FP Tree

In order to make the paper self-contained, we must first state the linear FP Tree algorithm that works on single core machines. Our aim is to parallelise this algorithm using fork/join methods so that it can run faster on tightly coupled multi-core machines.

2.1 FP-Tree Algorithm

FP-Growth [1] is an implementation of divide and conquer mechanism. It is done in two phases. In phase 1, the support of each word is determined. Then these are sorted, in order to get an F-Table. Now in the next phase, an FP tree is constructed using the entries and the F-Table. Then a threshold is applied in order to get a condensed FP-Tree. The algorithm performs mining recursively on FP-tree. The problem of finding frequent keywords is converted to searching and constructing trees recursively.

TID	Items
1	{a,b}
2	{b,c,d}
3	{a,c,d,e}
4	{a,d,e}
5	{a,b,c}
6	{a,b,c,d}
7	{a}
8	{a,b,c}

Fig.1.Transactional Database

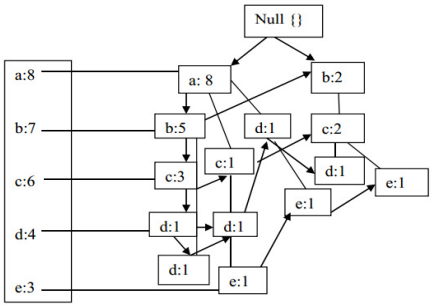

Fig.2.FP Tree with F-Table (left)

In the given example of Figure 1 we have 10 transactions, which are composed of alphabets from a to e. We can see that most items are of 3 elements, but it also contains items with only 1 element. This emphasises on the variation available in the data. In our scenario, of news data, there might be many news articles which are very small, and some might be huge. This is shown by the variation in the items.
Now the way in which we perform the FP tree algorithm can be seen by the steps given below**:**

Step 1: Create an empty F-List array F[]
Step 2: For every item in every transaction of the database, set F[item] +=1
Step 3: Sort the array F
Step 4: Create an empty tree T with null as the root node
Step 5: For every transaction in the Database, sort the transaction according to F-List, and add the items one by one to the tree T
Step 6: Maintain a reference of the items in the tree, and if present in more than one location, keep all the references.
Step 7: Starting with the leaf node, construct the conditional FP-Tree for each element.
Step 8: Generate the frequent Item Sets.

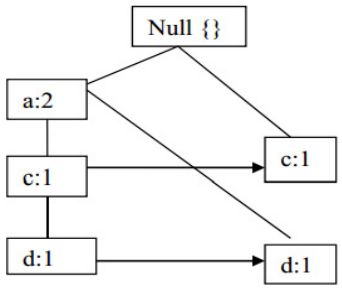

Fig.3. Conditional FP tree for e

Suffix	Frequent item sets
e	{e}, {d,e},{a,d,e},{c,e},{a,e}
d	{d}, {c,d},{b,c,d}, {b,d}, {a,b,d},{a,d}
c	{c}, {b,c}, {a,b,c},{a,c}
b	{b} {a,b}
a	{a}

Fig.4. Frequent item sets for each item

Hence the sorted F-list can be seen in Figure 2, along with the FP Tree. This tells us the relative abundance of the elements and the relation between various elements. A simple count measure can tell the most frequent item in the trend, but it would not be able to filter out the other unrelated topics. This can be done via the FP Tree. Figure 3 illustrates the conditional FP Tree that is generated for element e, this is done by looking out for all the parents of element e, till we do not reach the root node. Figure 4 shows a list of all elements with their item sets.

3. Proposed Improvements

We suggest 2 improvements to the above method. The first one is for operation in multi-core environments, where there exists a tightly coupled system with shared memory and shared resources, while the second one deals with our scenario of internet based trend analysis. The entire FP Tree is not needed for finding the Trends in the News world.

3.1 Parallelization

This technique works brilliantly on single core machines, but on multi core machines, it fails to utilize all the processing power available on modern multi-core machines. Each multi-core machine has its own way to schedule the tasks assigned to it, and may remain ideal if the tasks cannot be classified as independent. Majority of the computation is done to build the tree, as each transaction needs to be sorted according to the f-list first, and then added into the tree. There are tweaks available but the computation overhead attached to them defeats the purpose of parallelizing the algorithm. Like by using the threading techniques, we can assign the independent tasks, and make it work more efficiently on multi-core machines, but the threads, once assigned to a core, cannot be changed, and hence longer threads may block one of the cores, making the others effectively useless.

We suggest the use of Fork/Join methods instead, which are not so rigid, as they make use of work stealing mechanisms, and hence utilize all the power available to get the task done quickly and efficiently. We modify the Step 5 in the above algorithm, to make it compatible with multi core systems as follows

Step 5:

- **a. Define the number of active cores N in the machines**
- **b. Split the database into N parts with comparable transactions**
- **c. Fork the task, and assign each part to an individual core**
- **d. Each core sorts the transaction, and creates a local FP Tree**
- **e. Join all the FP Tree using intra-level Joining, Hence making a complete tree T**

Let us assume that N=2, hence the database is split into two parts, i.e. from entries 1 to 5, and remaining. The first part is given to the core 1, and the other part is given to core 2 of a dual core system. Now both the core work independently and give us their respective FP Trees, which can be seen in figure 5 (a) and (b). Now to join these partial Trees, we use Breath First Search, in which all the Nodes in the same level, are joined, and aggregated. The structure of the FP Tree is such that the items only occur in the same level, even when generated in parallel. If the Load is un-evenly balanced, on one core, other core can come in, and make the Tree for that core.

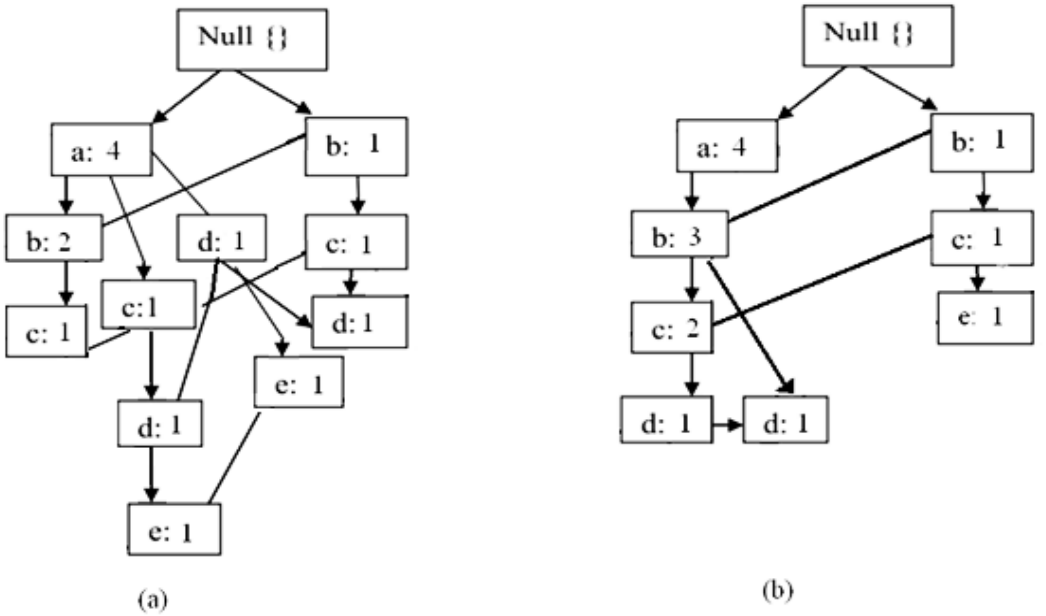

Fig. 5. (a) Partial FP Tree from core 1 (b) Partial FP Tree from core 2

3.2 Frequent keywords:

From the given FP Tree, we can find the frequent itemset in the database, but we do not need associations, instead dissimilarity. Hence we try to eliminate the small child nodes in the tree, and only consider the elements in the upper half of the tree. This enables us to increase the threshold for support in making the F-List. For example, say in the above example, The item ***a*** is trending, so are item ***c, d***, but items ***c, d*** are trending because of item ***a,*** without item ***a , c and d*** might not be trending, but on the other side, item ***b*** is also trending with item ***a*** and without it as well.

Hence the Trend in the transaction is that items a and ***b*** are the most frequent.

4. Application in Trend Analysis

We can use the above model to find out the most frequently occurring keywords in the news feed in order to analyze the data. Below is a Figure, which contains the headlines from the news. This feed is then converted into Keywords, which are processed using Natural Language processing (NLP) to obtain the unique Keywords in the feed. These Keywords act as the Items in an Item Set, and can be associated with an id like in the case of the transactional Database.

News feed	Keywords
OnePlus One Phones Sold in India Won't Receive OTA Updates: Cyanogen	OnePlus One, India, Cyanogen
OnePlus One is a Bit Too Big for a Phone	OnePlus One, Big Phone
No updates for Indian version of OnePlus One: Cyanogen	No updates, OnePlus One, Cyanogen
Cyanogen backpedals on promise to update OnePlus One phones in India	Cyanogen, backpedals, OnePlus One, India
OnePlus One bought via Amazon India won't get OTA updates, Cyanogen says	OnePlus One, India, Cyanogen
Cyanogen now says OnePlus One devices sold in India won't get updates	Cyanogen, OnePlus One, India
Cyanogen, Inc. looks to clarify OnePlus One update situation in India	Cyanogen, OnePlus One, India
Cyanogen says it will provide updates for OnePlus One in India	Cyanogen, OnePlus One, India
India Questions Roger Federer and Pete Sampras	India, Roger Federer, Pete Sampras
Roger Federer mesmerises Delhi	Roger Federer, Delhi
Rodger Federer lent, in his mostly unseen, often effortless way, a certain magic to the events here. There was always a certain elitism with tennis spectatorship in the Capital.	Roger Federer, tennis, the Capital.
I am still chasing perfection: Roger Federer	perfection, Roger Federer
Roger Federer says new IPTL event is 'crazy but fun'	Roger Federer, IPTL event
Roger Federer Gets Taste of India With 'Naan', Curry	Roger Federer, India, Naan, Curry
NCP and Congress join hands on drought	NCP, Congress, drought
The Congress and the NCP came together for the first time since the assembly election to take up the related issues of drought and farmer suicides in the state on Tuesday.	Congress, NCP, drought, farmer suicides
Vaiko Calls on EU, Bats for Referendum in Lanka	Vaiko, EU, Bats, Lanka
Goods worth over Rs 1.01 cr traded across LoC in Poonch	Goods, Rs 1.01 cr, LoC, Poonch
J&K assembly polls: Uri replies to terrorists with 79% turnout	J&K, assembly polls, terrorists
Netaji files: Centre told to file affidavit	Netaji files, Centre, file affidavit

Fig.6. Input data

Now following the steps of the algorithm explained above, we have to create an F List, or frequency list for the data. We have to put a Threshold to this list, and obtain just the keywords that occur more often. Failing to do so, may increase the breadth of the Tree unnecessarily, and make the Mining Unreliable. Figure 7 shows the common F list that is going to be used by both the cores.

OnePlus One	8
India	7
Cyanogen	6
Roger Federer	6
Congress	2
NCP	2
drought	2

Fig.7. F-List with Threshold set as 2

Next, we give 5 transactions to alternating cores, keeping the load balanced between the two cores. The Cores perform their local computations, to give us their respective local FP Trees.

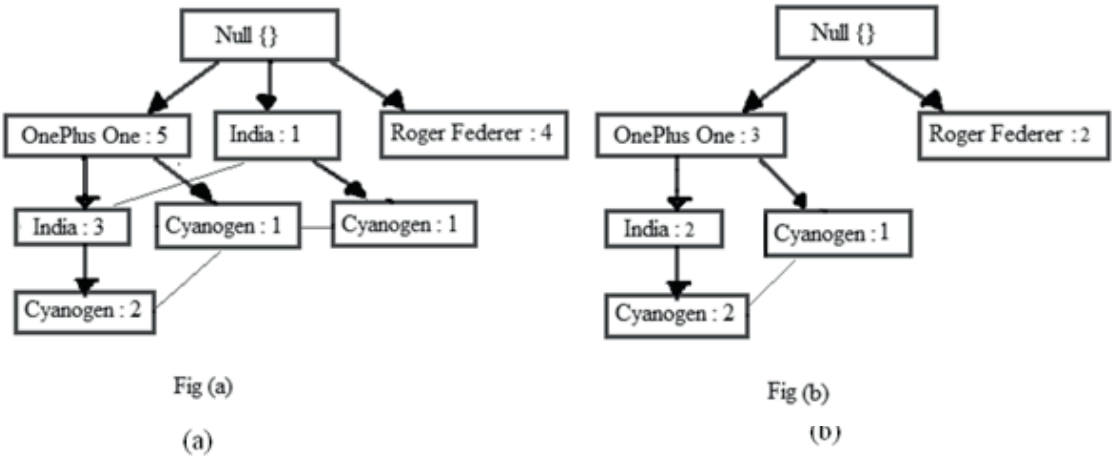

Fig.8. (a) Partial FP Tree from core 1 (b) Partial FP Tree from core 2

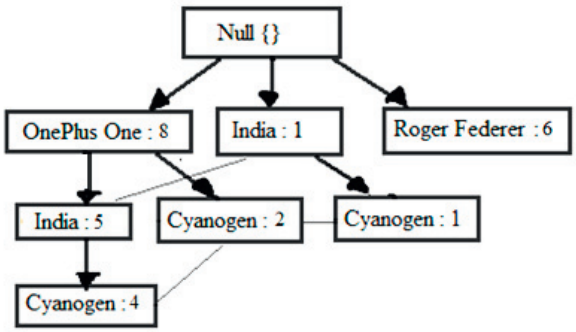

Fig.9: FP Tree for the given keywords

The final FP Tree after combining both the partial FP Trees from the individual cores are given in Figure 9.

.

The trending topics are ‘Oneplus One’, ‘Roger Federer’ and ‘India’.

5. Conclusion

As stated in the application, in which we use the FP Tree in order to generate the trends from a form of transactional database, instead of using FP Tree as a measure of generation of frequent itemsets, we use it in order to find out the relatively less frequent items in the database, but are unique and are not related to the main Trending topic of the database. As seen in the example above, comparing figure 7, to figure 9, In figure 7, ‘Roger Federer’ was the 4th most frequent topic, but in the FP Tree, he is placed in the second position in the level 1 items. In this way, ‘OnePlus One’, and it’s related Keywords were not able to over shadow ‘Roger Federer’. This was one of the examples, with a small data set of only 20 feeds. In the real world scenario, there may be multiple feeds that need to be sorted dynamically. As the method is modified for multi core systems, it can be run efficiently on the modern machines, which have limitless potential in them, but are most of the time ideal, not using all its power.

References

1. 1 J. Han and. M. Kamber *Data Mining: Concepts and Techniques* ISBN 13: 978-1-55860-901-3
2. H. Li, Y. Wang, D. Zhang, M. Zhang, E. Chang *PFP: Parallel FP-Growth for Query Recommendation* Proceedings of the 2008 ACM conference on Recommender systems Pages 107-114
3. S Sakurai *An Efficient Discovery Method of Patterns from Transactions with their Classes* Advanced Information Networking and Applications Workshops (WAINA), 2012 26th International Conference on 26-29 March 2012 Page(s):950 - 955
4. Seema L. Vandure, M. Ramannavar and N. S.Sidnal *Trend Projection using Predictive Analytics* International Journal of Computer Applications (0975 – 8887) Volume 97– No.19, July 2014
5. Doug Lea *A Java Fork/Join Framework* JAVA '00 Proceedings of the ACM 2000 conference on Java Grande Pages 36-43
6. Li Liu, Eric Li, Yimin Zhang, and Zhizhong Tang. *Optimization of frequent itemset mining on multiple-core processor*. In VLDB, 2007.
7. Muhaimenul Adnan and Reda Alhajj *A Bounded and Adaptive Memory-Based Approach to Mine Frequent Patterns From Very Large Database*s IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS—PART B: CYBERNETICS, VOL. 41, NO. 1, FEBRUARY 2011