DOI: http://dx.doi.org/10.26483/ijarcs.v9i5.6323

Volume 9, No. 5, September-October 2018

ISSN No. 0976-5697

International Journal of Advanced Research in Computer Science

RESEARCH PAPER

Available Online at www.ijarcs.info

A COMPARATIVE STUDY ON VARIOUS PARALLEL COMPUTING TECHNIQUES USING APRIORI ALGORITHM

Harshavardhan Metla
School of Computer Science and Engineering
Vellore Institute of Technology, Vellore, India

Yeshwanth Kamisetty
School of Computer Science and Engineering
Vellore Institute of Technology, Vellore, India

Sai Kiran Chintalapudi
School of Computer Science and Engineering
Vellore Institute of Technology
Vellore, India

Nalluri Rahul
School of Computer Science and Engineering
Vellore Institute of Technology
Vellore, India

Manikandan K
School of Computer Science and Engineering
Vellore Institute of Technology
Vellore, India

Siddharth Kolagatla
School of Computer Science and Engineering
Vellore Institute of Technology
Vellore, India

Abstract— A popular Association Rule Mining algorithm called Apriori algorithm helps in finding various frequent itemsets in the database. The constraints for finding these itemsets are given by the user in terms of support - measured by the proportion of transactions in which an itemset appears, and confidence - measured by the proportion of transactions with an itemset, in which another itemset also appears. The problem with this algorithm is that it is highly iterative and thus its efficiency rapidly decreases with increase in size or dimension of the dataset. Our project increases its efficiency with the help of openMP threads. We use data decomposition to split the transaction database into various parts, each taken by a thread to find the support count of all the candidate itemsets for all the transactions assigned to that particular thread. To give an example of the application, this project is used to determine the probability of the occurrence of a forest fire. Here, the transaction database can consist of various occurrences of natural phenomena, in which a few transactions also have the forest fire phenomenon, which means that it has occurred in the presence of the other itemsets in the transaction. Hence, if a new transaction is taken from the user, then the probability (or confidence) that a forest fire occurs, given this transaction, is calculated.

Keywords—Apriori, ARM, Forest, Itemsets, Mining, Parallelization, Probability, Transaction

I. INTRODUCTION

This is the era of data. There are huge amounts of data being generated every day. Data mining deals with the mining or the understanding of these data and deriving useful information from it. A common way of inferring something from the data is to understand the relationship between the different instances in the data or tuples in the table. This particular form of data mining is called Association Rules Mining (ARM), where associations between the different itemsets in the database are found with the help of various algorithms. A popular ARM algorithm is called Apriori algorithm. This algorithm helps in finding various frequent itemsets in the database. The constraints for finding these itemsets are given by the user in terms of support - measured by the proportion of transactions in which an itemset appears, and confidence - measured by the proportion of transactions with an itemset, in which another itemset also appears. The problem with this algorithm is that it is highly iterative and thus its efficiency rapidly decreases with increase in size or dimension of the dataset. Our project increases it's efficiency with the help of openMP threads. We use data decomposition to split the transaction database into various parts, each taken by a thread to find the support count of all the candidate itemsets for all the transactions assigned to that particular thread.

II. LITERATURE SURVEY

Apriori is an algorithm to learn the association rules among various transactions in a database. With the recent explosion of data, datasets are usually large with huge dimensions. [1] But Apriori traditionally doesn't perform well on datasets with large dimensions. So, this paper concentrates on reducing the dimensions of datasets using QR decomposition method, which calculated the similarity between two dimensions using their dot product.

Association Rule Mining is the mining of datasets to get information about the item sets that occur frequently in the dataset and Apriori is a widely used algorithm. [2] But this algorithm has less efficiency when used on large datasets. So, this paper focuses on using hash function to divide the dataset into buckets for easier calculation. It also introduces an idea for top- down implementation of the Apriori algorithm by pruning duplicate datasets.

The Apriori algorithm is made parallel with the help of Map reduce framework. [3] It normally consists of a map function and a reduce function. The map function takes an input from the data set and generates intermediate key pairs as output. The reduce function receives the output from the map function and collates together these values to form a set of values that are smaller than the original using an iterator. It outputs a new list of values. Since the map function accepts one input, all map operations are independent of each other and completely parallelizable. In the same way, reduce function can be made to run in parallel on each set of intermediate pairs with the same key.

This paper improves the efficiency of Aprori algorithm by using an important method known as "Cutting data mining". [4] This paper analyzes the Apriori algorithm for association rules mining, and make some improvement in the algorithm based on the features of cutting database.

Frequently occurring items in a large data set is really a time taking process. [5] So, Apriori algorithm is used to find the frequent pattern in itemsets. Apriori algorithm is parallelized using java concurrency libraries in multi core processors. Java concurrency framework is originally a library which enables us to perform parallel processing in multi core processors. This framework provides then multi-threading feature where various threads are executed simultaneously by writing concurrent applications. Both the serial and parallel code performance can be measured on multicore processors by calculating the time.

In order to handle the high Input/Output time, a work flow model called Spark has been introduced. [6] This approach exceeds the Map Reduce implementation of Hadoop in both efficiency and performance. The R-apriori uses both Map Reduce and Spark for converting the normal Apriori algorithm to parallel. The R-apriori is the most efficient algorithm which overcomes the Spark apriori and Map Reduce Apriori on Hadoop in speedup and lower execution of time.

In this method GPU is also run along with the CPU to decrease the execution time of the algorithm, this is called GPU accelerated computing. [7] Apriori algorithm is parallelized using on Graphic Processing Unit. Speed up of Apriori algorithm is measured in both sequential and parallel execution if the code. Some part of the code which can be executed in parallel uses GPU with CUDA parallel architecture.

This paper brings a more efficient algorithm after the analysis of classical Apriori algorithm. [8] By examining the database only once, all the transactions are refactored into the components of a two-dimensional array. The calculation turns out to be more accurate by introducing weight. In addition to this, the deletion of redundant data, joining and pruning steps become simple. Therefore, this enhances the efficiency of the Apriori algorithm.

This algorithm performs a new method to decrease the repetitiveness of sub-things during pruning the competitor itemsets. [9] This can shape straightforwardly and directly the arrangement of successive itemsets and take out the competitor having a subset that is not visited. This algorithm can increase the probability of retrieving information in scanning the database and also reduce the potential scale of itemsets.

Apriori algorithm is parallelized using openMP and multicore-processors (Quad core). [10] The main objective is to measure the performance of Apriori algorithm in serial and parallel execution using openMP on a quad core processor. OpenMP is an API (Application Programming Interface), which uses fork and join model for performing parallelism. In this model, some part of the program is sequentially executed in master thread and the other part that is parallel is run on child threads. Master thread creates a group of child threads to run the parallel part. The work which is shared equally among the child threads are synchronized and evaluated.

Traditional methods for forest fire forecasting by applying the weather data can only be used as a prediction for an administrative division or province level, and it does not favour the farmers who are mainly affected, because there is no accurate forest fire prediction due to lack of forest fire monitoring. [11] Hence, Apriori, an association rule algorithm, is applied to analyze the probability and intensity of a forest fire effectively with coarse forest fire data. It helps the farmers at the ground level to predict the forest fire applying this coarse weather data.

III. EXISTING METHODOLOGY

Serial Code for Apriori Algorithm

A frequent itemset is an itemset whose support count is greater than the user specified minimum support count. (L_k where k is the size of the itemset). A candidate itemset is an itemset from where frequent itemset is extracted. In Apriori algorithm, initially a candidate item set of size k=1 is created and from them a frequent itemset is generated by pruning the candidates with the support count less than given support count.

The candidate item set of size k is generated from the frequent itemset of size k-1. If there are two frequency itemsets p and q of size k-1 then the candidate itemset is generated if all the items of p and q match except the last one. This is done for all the possible pairs of frequent itemsets of size k-1. Now for these itemsets, pruning is done. If any subset of size k-1 in any of these sets is not in the frequency itemsets then that set if removed. After this the database is scanned to find the support of each of these candidate sets and the ones with the support greater than threshold are put in the frequent item set L_k.

This algorithm is an iterative algorithm and at every next step the frequency itemset of size k is generated. The size of k increases by 1 in every iteration. This algorithm goes on until no frequent itemset of size k is found.

- **void C1(string file_name)** - This function generates the first candidate list.
- **void output(structure T)** - It prints all the candidates or the frequency list along with their count.
- **void L1()** - it generates the first frequency list.
- **void generate_C()** - It generates all the candidates of size k from frequency list of size k-1.
- **bool check_compatibility(VI a,VI b)** - This function checks if the two frequency itemsets are same or not.
- **void prune()** - It removes all the candidates of size k whose subsets are not present in the frequent itemsets of size k-1.

- **void scan_D(string file_name)** - It scans the database and calculates set_count, i.e, calculating the support count for each transaction.
- **void set_count(VI a)** - It increments the support count of the candidates if exists in the database transaction. it is done for all the candidates.
- **void generate_L()** - It finds the frequent itemsets of size k by removing all the candidate item sets whose support count is less than the min support count.

IV. PROPOSED METHODOLOGY

A. Parallelizing the Serial Code for Speedup

Since our implementation of Apriori algorithm include 9 functions, it is very important to recognize the part of the code to be parallelized. The Apriori algorithm is itself an iterative algorithm and each iteration depends on the values obtained by the previous one, i.e. there exists a real loop dependency between the iterations. Since each iteration take significant amount of time, each iteration needs to be parallelized. From the previous studies, it has been found that about 90% of the time is consumed in scanning the transactions in the database to find out whether a particular candidate is present in the transaction or not. The function which scans the database if scan_D(). It also calls set_count to calculate the support for each candidate which also contributes to the large amount of the time spent.

We use data decomposition technique to divide our transaction data into the number of threads. Since the transactions are divided into the number of threads, now for every candidate the scanning and calculating the set_count is done parallelly by different threads and leading toward the speedup. After the set_count is calculated by different threads for each candidate, all the set_counts for each candidate are added and hence final set_count for every candidate is found. These counts for all the threads are added in the critical section.

The critical section in the program is implemented using Barrier Synchronization. barrier synchronization has been used to add the calculated set_count values and find the final support for each candidate. A barrier for a group of threads in a section of code means that any thread /process must stop at this point and cannot proceed until all other threads/processes reach this barrier. So, in our code, until all the threads calculate the set_count for each candidate, the summation of set_counts cannot be done.

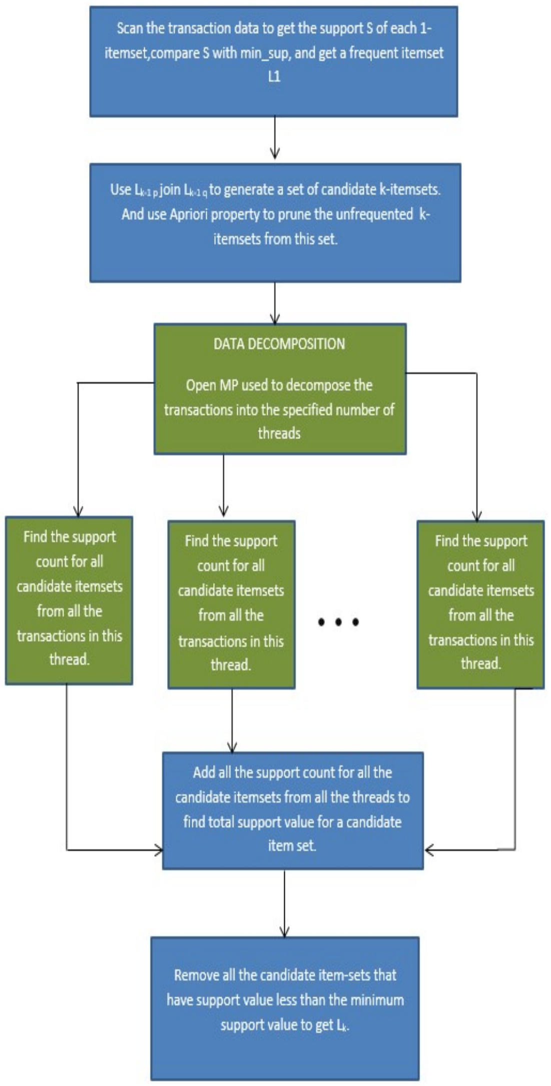
B. Forest Fire Application

As an application to Apriori algorithm, we have chosen to detect whether the forest fire will occur after a certain set of events has already occurred. Our database consists of the numbers indicating different events occurring the forest. Number 12 indicates the occurrence of a forest fire.

We, hereby calculate the confidence score as the probability of occurrence of forest fire after certain events have happened. So, if the new transaction is called T, then,

$$Probability\ (Forest\ Fire) - \frac{\sup(T,12)}{\sup(T)}$$

The function used in our code for this purpose is:
float predict(string input) - This function uses the structure F which collects all the levels that have been generated in the Apriori algorithm. Now, this function takes the input transaction as a parameter and converts it into a vector<int> template by converting each individual element in the transaction from a character to an integer. The function then finds the count of this transaction from the structure F. This is the support/frequency/count of the transaction. Now, the support of the transaction along with the number 12, which mimics the occurrence of a forest is found out from the same structure F. Now, the two support values that we have will give us the confidence (or probability) that 12 will occur after the given transaction. This is done by dividing the former support count by the later support count.

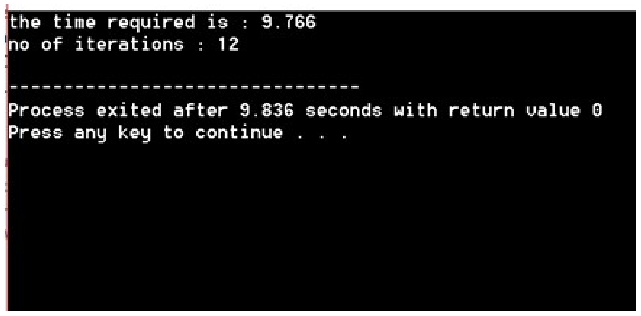

Scan the transaction data to get the support S of each 1-itemset,compare S with min_sup, and get a frequent itemset L1
Use $L_{k-1\,p}$ join $L_{k-1\,q}$ to generate a set of candidate k-itemsets. And use Apriori property to prune the unfrequented k-itemsets from this set.
DATA DECOMPOSITION
Open MP used to decompose the transactions into the specified number of threads
Find the support count for all candidate itemsets from all the transactions in this thread.
Find the support count for all candidate itemsets from all the transactions in this thread.
• • •
Find the support count for all candidate itemsets from all the transactions in this thread.
Add all the support count for all the candidate itemsets from all the threads to find total support value for a candidate item set.
Remove all the candidate item-sets that have support value less than the minimum support value to get L_k.

Flow of execution

V. RESULTS AND DISCUSSION

A. Serial

Here, the original Apriori algorithm was implemented in C++ and the time of execution was calculated using the omp_get_wtime() function from the <omp.h> header file. The program starts of by calculating the first candidate list by reading the text file and counting the support value of each distinct item in the database. This forms the first candidate list of the transactions. This candidate list is then filtered against the MIN_SUP value, i.e., the minimum support value that can be accepted. This forms the first level of the Apriori algorithm. The items of this level are then joined to make a candidate list for the next level (2 items in an itemset). First this candidate set is pruned by the Apriori algorithm. This removes all the itemsets whose subsets aren't frequent itemsets. Now, the count of the rest of the itemsets are found by going through the transactions database. Once the count for each of the itemset is found, the new candidate set is filtered against the MIN_SUP value and the new level of the apriori algorithm is found. This process is repeated for all the possible levels that can be formed from the items in the given transaction.

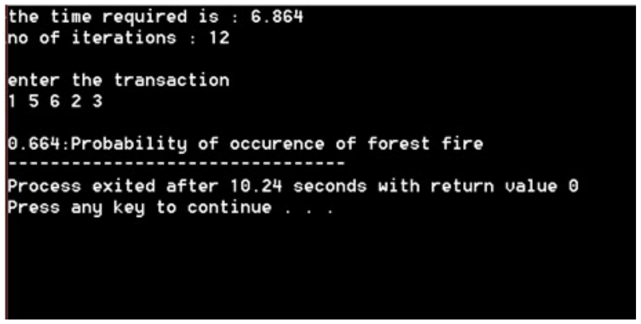


```
the time required is : 9.766
no of iterations : 12

--------------------------------
Process exited after 9.836 seconds with return value 0
Press any key to continue . . .
```

B. Parallel

To overcome the need of the Apriori algorithm to perform an impractically large number of iterations over the database every time it needs to generate the candidate list, we have implemented it in a parallel style using the OpenMP library in C++. Whenever the algorithm needs to scan the database to find the support of every element in the candidate list, it performs a read of the database. This is the main portion of repetitiveness in the algorithm. Hence, this portion of the code has been parallelized. Here, data decomposition is performed by splitting the database into the number of threads that are running. Each thread starts at a particular point in the file and reads till another point where the other thread would have started from. Each of these threads finds the support count of the itemsets, in the candidate list that has called this function. These counts are then summed up through a barrier OpenMP directive. This method has shown to significantly reduce the time taken to perform the apriori algorithm. To show an application of this algorithm it has been implemented to predict the occurrence of forest fires, more about this is dealt with the next section.



```
the time required is : 6.864
no of iterations : 12

enter the transaction
1 5 6 2 3

0.664:Probability of occurence of forest fire
--------------------------------
Process exited after 10.24 seconds with return value 0
Press any key to continue . . .
```

C. Forest Fire Application

To see how the Apriori algorithm works in a real-world application, it is implemented to forest fire prediction. Here each transaction in the database is considered to consist a set of items each of which mimic a particular natural phenomenon. So, here the item 12 is considered to be the event that a forest fire occurs. If a new transaction is entered by the user, then the probability the occurrence of 12 along with it is calculated and this is shown as the probability of the occurrence of a forest fire, given that all the natural phenomena given to the algorithm in the form of a transaction.

VI. CONCLUSION AND FUTURE IMPLEMENTATION

Thus, the parallel implementation of the Apriori algorithm reduces the time taken for the algorithm to scan the database. This increases the efficiency of the algorithm and also makes it more usable for real time applications.

The Forest Fire prediction application is one such real time application where the apriori algorithm is used to predict the occurrence of a forest fire.

The parallel implementation of the algorithm is just the first step in increasing the efficiency. Further down the road, a data structure can be implemented to retrieve the entire dataset from the database and store it in the memory. This would drastically decrease the time as the whole process of reading the database is eliminated. Also, better pruning methods can be used to remove the unwanted itemsets from the candidate list before forming the Apriori level.

Till the discussion was limited to the basic Apriori algorithm. But a lot of variations of the algorithm have been made and there are other frequent data mining algorithms also present. Some of which perform better than the original Apriori algorithm in a few cases. Even these algorithms can be worked on.

REFERENCES

[1] Hu, L., Zhuo, G., & Qiu, Y. (2009, August). Application of Apriori algorithm to the data mining of the wildfire. In Fuzzy Systems and Knowledge Discovery, 2009. FSKD'09. Sixth International Conference on (Vol. 2, pp. 426-429). IEEE.

[2] Korde, N. S., & Shende, S. W. (2014). Parallel Implementation of Apriori Algorithm. IOSR Journal of Computer Science, 01-04.

[3] Chai, S., Yang, J., & Cheng, Y. (2007, June). The research of improved apriori algorithm for mining association rules. In Service Systems and Service Management, 2007 International Conference on (pp. 1-4). IEEE.

[4] Changsheng, Z., Zhongyue, L., & Dongsong, Z. (2009, March). An improved algorithm for apriori. In Education Technology and Computer Science, 2009. ETCS'09. First International Workshop on (Vol. 1, pp. 995-998). IEEE.

[5] Spandana, K., Sirisha, D., & Shahida, S. (2016). Parallelizing Apriori Algorithm on GPU. International Journal of Computer Applications, 155(10).

[6] Rathee, S., Kaul, M., & Kashyap, A. (2015, October). R-Apriori: an efficient apriori based algorithm on spark. In Proceedings of the 8th Workshop on Ph. D. Workshop in Information and Knowledge Management (pp. 27-34). ACM.

[7] Parsania, V., Kamani, G., & Ghodasara, Y. R. (2014). Mining Frequent Itemset Using Parallel Computing Apriori Algorithm.

[8] Wang, G., Yu, X., Peng, D., Cui, Y., & Li, Q. (2010, June). Research of data mining based on Apriori algorithm in cutting database. In Mechanic Automation and Control Engineering (MACE), 2010 International Conference on (pp. 3765-3768). IEEE.

[9] Li, N., Zeng, L., He, Q., & Shi, Z. (2012, August). Parallel implementation of apriori algorithm based on mapreduce. In Software Engineering, Artificial Intelligence, Networking and Parallel & Distributed Computing (SNPD), 2012 13th ACIS International Conference on (pp. 236-241). IEEE.

[10] Shah, A. (2016, July). Association rule mining with modified apriori algorithm using top down approach. In Applied and Theoretical Computing and Communication Technology (iCATccT), 2016 2nd International Conference on (pp. 747-752). IEEE.

[11] Harikumar, S., & Dilipkumar, D. U. (2016, August). Apriori algorithm for association rule mining in high dimensional data. In Data Science and Engineering (ICDSE), 2016 International Conference on (pp. 1-6). IEEE.