SURVEY ON PARALLEL ALGORITHMS IN ASSOCIATION RULE MINING

Pradeep chandra
20161145

P.Preetham
20161063

ABSTRACT

Finding frequent item sets is one of the most important field of research in data mining . Several research has been done on this topic since long time . Of all the algorithms that are present, the Apriori algorithm and the FP growth algorithms have been improvised, implemented and evaluated several times . Both these algorithms were well defined and widely accepted in the association mining tasks. Several research papers were published improvising the time complexity of these algorithms over the decade since the first time these algorithms were written . Among many techniques that were used to improvise these algorithms parallelization is one of the important technique used to improve them . Parallelization of these algorithms has reduced the time complexity of these algorithms by a significant amount . In this paper we are presenting a survey on how parallelizing has reduced the time complexity of these algorithms by comparing the general algorithms with their parallelized counterparts .

SECTION 1.

INTRODUCTION

Finding frequent itemsets is very important for many business purposes . Among many algorithms that were present, most of them are implemented based on apriori and frequent pattern growth algorithms . parallelization is one of the important technique that can be used to improve these algorithms . The Apriori algorithm is required to generate candidate itemsets, compute the support, and prune the candidate itemsets to the frequent itemsets in each iteration . On the other hand FP growth algorithm doesn't require the expensive step of generating the candidate keys . In many cases FP growth algorithm gives results in less time when compared to the apriori algorithm which requires generating the candidate keys in every iteration .

In this paper we will explain how applying parallelization on these existing apriori and FP growth algorithms improve their time complexities to get their results . Since we have already declared that the FP growth performs better than apriori algorithm , we are not comparing these algorithms in the preceding sections .The rest of the paper is organized as follows. Section 2 introduces about the general apriori algorithm . Section 3 presents how parallelization can be applied on the apriori algo and its advantage over general apriori algorithm . Section 4 explains FP growth algorithm and Section 5 explains about parllelization of the algorithm .

SECTION 2.

APRIORI ALGORITHM

Let D be a database of transactions, where each transaction has a unique identifier (tid) and contains a set of items called an itemset . An itemset with k items is called a k-itemset . A k-subset is a k-length subset of an itemset .An itemset is frequent or large if

its support is more than a user-specified minimum support (min_sup) value .

The apriori algorithms need minimum support to be specified to find out the candidate 1 itemsets at the starting of the algorithm . The algorithm then generates the frequent 1-itemsets by pruning some candidate 1-itemsets if their support values are lower than the minimum support. After the algorithm finds all the frequent 1-itemsets, it joins the frequent 1-itemsets with each other to construct the candidate 2-itemsets and prune some infrequent itemsets from the candidate 2-itemsets to create the frequent 2-itemsets. This process is repeated until no more candidate itemsets can be created .

Thus in every iteration of apriori algorithm first step includes the pruning of the infrequent candidate itemsets from the candidate set generated in previous iteration to get frequent itemsets . The second step involves generating of the candidate itemsets from the frequent itemsets obtained in the first step . This continues for several iterations still there were no candidate itemsets available .

Candidate Generation

Given L(k-1) , set of all frequent k-1 itemsets , we want to generate a superset of the set of all frequent k-itemsets .The intuition behind the Apriori candidate generation procedure is that if an itemset X has minimum support, so do all subsets of X.For simplicity, assume the items in each itemset are in lexicographic order .

Candidate generation is done in two steps . First step join any itemset in La(k-1) with Lb(k-1) if and only if all the first (k-2) items in both the itemsets are same , when arranged in the lexicographical order .Then consider lexicographically smaller items as (k-1)th item of the new k-itemset formed , among the (k-1)th item of A and B .

After forming all the itemsets like this remove the itemsets if any of their subset of length k-1 is not present in L(k-1) .

SECTION 3.

PARALLEL IMPLEMENTATION

Parallelism can be of two types.

1. Data parallelism
2. Task parallelism

Data parallelism means, the case where the whole database is divided into several parts and each part is assigned to a processor . Each processor then work on their own database to find the local count which are then used to find the global count for the candidate set .

In Task parallelism each database computes their own candidate sets which are disjoint. But in task parallelism each processor requires the access to the whole database .

There also exist a Hybrid parallelism which combines both task and data parallelism , which is the important parallelism technique to exploit the all available parallelism .

Three types of parallel algorithms were given in the research paper written by Rakesh Agarwal and John Shafer .Their target machine was a 32-node IBM SP2 DMM . They have applied these techniques on the apriori algorithm so the steps in these algorithms will be self explanatory as it was discussed above . These three algorithms were

1. Count Distribution
2. Data Distribution
3. Candidate Distribution

These algorithms assume a shared nothing architecture , where each of N processors has their own private memory and a disk .

The processors are connected by a communication network and can only communicate by passing messages .

Count Distribution

Let P1, P2, P3 , ... ,Pn be the processors .The algorithms first divides the the whole data set D into n pieces and assigns each processor Pi a database Di .

1. During the algorithm each processor generates the whole candidate itemset at every iteration .
2. Then each processor Pi scans their local database Di and develops a local support count for candidates in candidate itemset generated above .
3. Then all processors communicate with each other to find the global support counts ,for the pruning process .
4. Each processor Pi uses the global support count obtained above to find the frequent itemsets .

Thus in Count Distribution algorithm every processor has to store the same candidate itemset . This means if the candidate set generated requires memory m and we have n processors then total memory required to run this algorithm will be O (nm) which is more space required than the sequential apriori . But the advantage of this algorithm is that every processor develops their local support counts based on Di's allocated to them and this counting will be done at the same time ie synchronously which reduces the time taken when compared to sequential algorithms as it has to scan the whole database at every iteration . But remember that these local counts has to be synchronized at every iteration to get the global count .

This algorithm minimizes communication, because only the counts are exchanged among the processors .However, because the algorithm replicates the entire candidate set on each processor, it doesn't use the aggregate system memory effectively.

DATA DISTRIBUTION

In the data distribution algorithm each processor counts mutually exclusive candidates .This algorithm uses total system memory by generating disjoint candidates sets on each processor .But for the purpose of pruning the candidate set to get frequent itemset each processor need to find the global support which means each processor has to scan the whole database to get their required global support for every iteration .To get the count , each processor has to find count in its own partition and also in all the remote partitions held by the other processors which means that to find the count that particular processor Pi has to get the data from other processors by sharing the data .So this algorithm requires lot of communication to be done between the processors which makes it less desirable algorithm when compared to the count distribution algorithm because in count distribution algorithms only the local count has to be communicated between the processors rather than communicating the frequent itemsets as in the case of data distribution .

CANDIDATE DISTRIBUTION

Both the above proposed algorithms require either count or itemsets to be distributed between the processors which adds some cost to the time at each iteration . Sometimes if the workload is not balanced among the processors then it may happen that all the processor has to wait for the last particular processor to complete its computation in order to move to next iteration .

In candidate distribution algorithm these mistakes were overcomed . The Candidate distribution algorithm attempts to do away with these dependencies by partitioning both the data and the candidates in such a way that each processor may proceed independently. In some pass L, where L is heuristically determined, this algorithm divides the frequent itemsets Ll-l between processors in such a way that a processor P' can generate a unique candidate itemsets independent of all other processors and the database D is repartitioned in such a way that this processor can also find the global count of these candidate itemsets from its database Di only independent of other processors .

The algorithms does the candidate partition at a iteration L .In the research paper the partition was done at fourth iteration . In each iteration , each processor asynchronously broadcasts its frequent itemset to all other processors . If this information arrives in time then it is used otherwise it will be saved for the next iteration .

Even though it reduces all the dependencies present in the previous algorithms it performs bad when compared to count distribution because at every iteration it has to repartition the whole database among the processors which takes significant amount of time at every iteration .

Takahiko Shintani and Masaru Kitsuregawa parallel algorithms

These two have independently given 4 algorithms . Among them three were quite similar to the above algorithms but the last one is different and they named it as HPA_ELD . The essence of this algorithms is that even we are dividing frequent itemsets we may divide them unevenly among the processors ie some processor may get high load and some may get lower load . This problem is solved by them in this paper and they have told that it outperforms all the three algorithms presented above .

SECTION 4.

SEQUENTIAL FP GROWTH ALGORITHM

As we know in apriori algorithm it is of very high time complexity due to repeated full scans. To reduce this we introduce fp-growth algorithm in which the scaning of datatable is done only twice.FP-Growth uses a pattern fragment growth method to avoid the large number of scans on the database.. FP-Growth algorithm goes as follows ,Consider a database A and T be the set of all transactions. Similar to apriori we mention a minimum support threshold. Consider all Transactions ti that belongs to T . the frequency of each item is calculated . Items which do not satisfy the mentioned threshold are neglected and based on the descending order of frequencies , each transaction is again sorted
for example consider a transaction ti {A , C,B Z, E G} and the list of all itemsets which satisfy the threshold in descending order be {A,B,C,D,E} then the transaction ti is converted to {A,B,C,E}. Then based on the modified transaction table FP-tree is built with weights .With a null root all the item sets are made as nodes with each path till leafnode depicting a transaction.
The above procedure is part1 of our algorithm that is FP-Tree Build()
For each frequent item, we construct its conditional pattern-base, and then its conditional FP-tree . Repeat the process on each newly created conditional FP-tree Until the resulting FP-tree is empty, or it contains only one path(a single path will generate all

the combinations of its sub-paths) each of which is a frequent pattern .

This the part2 of the algorithm ie., FP-growth() .Thus generating frequent patterns which in turn are used in generating ARs, this is what constitutes the majority of runtime and thus needs to more optimized for better performance
We can clearly see that a divide and conquer strategy is used with top down approach,with the advantages being mainly 3 , those are no candidate generation ,compressed FP-Tree data structure and no repeated scans .

SECTION 5.

PARALLELIZATION OF FP GROWTH ALGORITHM

Instead of being a lot faster than apriori, the fp-growth can be memory wise inefficient as the its main data structure used can be quite large for very large datasets and computational costs would be very high .In order to reduce the drawbacks of sequential FP-Tree algorithm a new mining algorithm MFPT has been proposed in [6] which is based on the FP-Tree algorithm (Multiple local frequent pattern tree algorithm) .

In this algorithm the dataset is divided into equal amount of data between the processors available and each processor builds its own fp-local tree using the part of data assigned to it .this phase requires a second complete i/o database scan .For each transaction read by a processor only the set of frequent items present in the header table is collected and sorted in descending order according to their frequency .Then again based on these local trees the frequent pattern are mined parallely . This paper proposed a solution for load balancing among processors and resource sharing with minimum mutual-exclusion locking ie., related to the modern architecture of shared memory and shared hard drive architecture .

Fp-Tree algorithm is observed to have 2 stages Fp-growth and Fp-build where Fp-growth takes up the majority of the time .In FP-growth, major part of the execution time is spent in the condition_pattern_base() routine, which finds the frequent items in the conditional pattern base for an item, and the rest of the execution time is spent in the FPGrowth() routine.In this paper [3] an idea to improve the FP-growth has been shown on the modern multicore processor architecture. The cache which is not generally properly used throughout the runtime of the algorithm has been made of some use by introducing cache-conscious frequent pattern array (FP-array), a data structure designed to significantly improve cache performance.A cache-conscious FP-array is a data reorganization of FPtree by transforming it into two arrays, i.e. item array and node array. The transformation of Fp-Tree has been done using depth-first order and stored in a reverse manner to maintain inorder traversal .Furtherer modification to the FP-array is made by dynamically choosing the node size ranging from 4 bytes to 1 byte in the item array according to the total frequent items in use.Data prefetching is made in order to reduce the cache miss ie., software and hardware prefetching. Even a cache conscious tree (CC-Tree)has been introduced but Fp-array is shown to be better in improving the cache -line utilization.

The main problem with fp-tree algorithm is because of its enormous size ,as each transaction corresponds to a particular part of the tree but the next transaction can have a completely different path ,so implies a complete cache miss. So, data tiling is done so that the transactions are ordered in such a way to reduce the cache miss and reuse the same portion of Fp-tree in a temporal fashion.Data tiling is done by firstly

classifying most frequent items to hot-items and rest to cold-items and building a hot-item tree based on this.The transactions which can be processed from one part of the hot-tree are grouped into one tile.All the data insertion in one tile only performs in one subset of the FP-tree,The procedure iterates until all the tiles complete transaction insertion.The advantages of this data-tiling is that it improves the temporal cache locality performance and lays a foundation for thread level parallelization.Because of the lock-based exclusive access mechanism to prevent concurrent access the parallel program runs slower than it actual sequential version.To rectify this an idea of Lock-free Fp-tree building has been introduced.In this idea as previously mentioned above , a tree is constructed based on hot-tree ,A dynamic scheduling algorithm is adopted to minimize the load imbalance.Since the transaction are inserted in the hot-tree in context of tile format and each tile corresponds to unique class , there is no overlap.Thus,significantly improving the scaling performance on the multi-core processor.

All the above improvements were shown to be very efficient and productive.Both [6] and [3] parallelized the FP-Growth algorithm in a shared memory system.

Further performance can be expected from parallel execution on shared nothing environment, such as a computer cluster.Thus a novel approach has been proposed in [9] ie., for parallel execution of Fp-growth .GFP-Growth adopts the projection method to find all the conditional pattern bases directly without constructing an FP-tree. It then splits the mining task into a group of independent sub-tasks, executes these sub-tasks in parallel on nodes and aggregates the results back into a final result.The process of finding all conditional pattern bases is sequentially executed on a single computer, which is called the master node.

The above algorithm main features are 1.subtle division of sub-tasks ,which are independent of other thus need minimum or no communication.2.Dynamic load balancing maximizes the use of computer resources efficiently,3.Asynchronous communication in which each non master node communicates with the master node.The run time has been shown to be significantly brought lower than Fp-growth with a better parallelization approach.

In the paper[4] though the main focus is on efficient query recommendation , the frequent patterns have been efficiently computed by parallelizing fp-growth algorithm.The PFP algorithm firstly division and distribution of data is done which is sharding , Then we use a mapreduce pass to effectively compute support values of all item sets in the database.then the items are grouped into group lists .Then in the actual PFP algorithm ,we convert transactions in DB into some new databases of group-dependent transactions so that local FP-trees built from different group-dependent transactions are independent during the recursive conditional FP-tree constructing process. The Mapper and Reducer procedures are assigned for this task.They use the MapReduce infrastructure.the MapReduce infrastructure collects corresponding group-dependent transactions as value value′ , and feed reducers by key-value pair.the support of a pattern can be counted only within the group-dependent shard with key′ = gid(groupId), but does not rely on any other shards.Then Fp-growth on Group dependent shards is done ,the reducer constructs the local FP tree and recursively builds its conditional sub-trees similar to the traditional FP-Growth algorithm, the patterns

are not output directly, but into a max-heap indexed by the support value of the found pattern.Then comes the aggregation step which reads the output from above steps and prints the output. For each item, it outputs corresponding top-K mostly supported patterns.This algorithm is latest algorithm with most efficient logical implementation with maximum effective parallel execution of above papers.

FUTURE WORK

We came across some new research papers especially on distributed systems and in the paper[7] there is a FIDOOP algorithm to cross check and some algorithms which use modern distributed systems architecture in much more efficient manner .

There are also some algorithms which have used the mapreduce to make these algorithms efficient , we are going to do a survey on these algorithms in future .

REFERENCES

[1] R.Agrawal and R.Srikant, ”Fast Algorithms for Mining Association Rules,” Proc.20th Int‘l Conf.Very Large Databases,Santiago,Chile, Sept.1994.

[2] R Agrawal and J. Shafer, ”Parallel Mining of Association Rules: Design,Implementation, and Experience,” Research Report RJ 10004, IBM Almaden Research Center,San Jose, Calif.,Feb. 1996.

[3]. Li Liu,Eric Li , Yimin Zhang , Zhizhong Tang:Optimization of Frequent Itemset Mining on Multiple-Core Processor .

[4]Haoyuan Li,Yi Wang,Dong Zhang.PFP: Parallel FP-Growth for Query Recommendation .

[5] Takahiko Shintani, Masaru Kitsuregawa . Hash based parallel algorithms for mining association rules . DIS '96 Proceedings of the fourth international conference on on Parallel and distributed information systems.

[6] Osmar R. ZaYane,Mohammad El-Hajj :Fast Parallel Association Rule Mining Without Candidacy Generation .

[7]Yaling Xun, Jifu Zhang, and Xiao Qin,: FiDoop: Parallel Mining of Frequent Itemsets Using MapReduce .

[8]Li Li ,Donghai Zhai ,Fan Jin : A Parallel Algorithm for Frequent Itemset Mining .

[9]Min Chen, XueDong Gao ,HuiFei Li :An Efficient Parallel FP-Growth Algorithm .