AN EFFICIENT PARALLELIZED ALGORITHM FOR MINING ASSOCIATION RULES

G.SUMATHI[#1]*,M.INDUMATHI*[*2]*,K.TAMILARASI*[#3]
[#]*ASSISTANT PROFESSOR, ME STUDENT, DEPT OF CSE,*
MUTHAYAMMAL ENGINEERING COLLEGE
RASIPURAM - 637 408, NAMAKKAL DT,INDIA.
subamecsep@gmail.com
indumathimurugesan@gmail.com
[*]*M.E STUDENT, DEPT OF CSE,*
MUTHAYAMMAL ENGINEERING COLLEGE,
RASIPURAM - 637 408, NAMAKKAL DT, INDIA.
Tamilarasik1990@gmail.com

ABSTRACT

A new distributed association rule mining (D-ARM) algorithm is presented in this paper that demonstrates super-linear speedup with the number of computing nodes. Our distributed algorithm scans the database once, just like the Sampling algorithm, and is thus more efficient than any D-ARM algorithm known today, not only this algorithm divide the disk-I/O costs of the single scan by partitioning the database among several machines, but also uses the combined memory to linearly increase the size of the sample. This increase further improves the performance of the presented, distributed association rule mining algorithm.

1. INTRODUCTION

The economic value of data mining is today well established. Most large organizations regularly practice data mining techniques. One of the most popular techniques is association rule mining (ARM), which is the automatic discovery of pairs of element sets that tend to appear together in a common context. An example would be to discover that the purchase of certain items in a supermarket transaction usually implies that another set of items is also bought in that same transaction. Like other data mining techniques that must process enormous databases, ARM is inherently disk-I/O intensive.

These I/O costs can be reduced in two ways: by reducing the number of times the database needs to be scanned, or through parallelization, by partitioning the database between several machines which then perform a distributed ARM (D-ARM) algorithm. In recent years much progress has been made in both directions. The main task of every ARM algorithm is to discover the sets of items that frequently appear together the frequent itemsets. The number of database scans required for the task has been reduced from a number equal to the size of the largest itemset in Apriori , to typically just a single scan in modern ARM algorithms such as Sampling and DIC Much progress has also been made in parallelized algorithms. With these, the architecture of the parallel system plays a key role.

For instance, many algorithms were proposed which take advantage of the fast interconnect, or the shared memory, of parallel computers. The latest development with these is, in which each process makes just two passes over its portion of the database. Parallel computers are, however, very costly. Hence, although these algorithms were shown to scale up to 128 processors, few organizations can afford to spend such resources on data mining. The alternative is distributed algorithms, which can be run on cheap clusters of standard, off-the-shelf PCs. Algorithms suitable for such systems include the CD and FDM algorithms , both parallelized versions of Apriori, which were published shortly after it was described.

However, while clusters may easily and cheaply be scaled to hundreds of machines, these algorithms were shown not to scale well. The DDM algorithm, which overcomes this scalability problem, was recently described. Unfortunately, all the D-ARM algorithms for share-nothing machines scan the database as many times as Apriori. Since many business databases contain large frequent itemsets, these algorithms are not competitive with DIC and Sampling. In this work we present a parallelized version of the Sampling algorithm, called P-Sampling. The algorithm is intended for clusters of share-nothing machines. The main obstacle of this parallelization, that of achieving a coherent view of the distributed sample at reasonable communication costs, was overcome using ideas taken from DDM. Our distributed algorithm scans the database once, just like the Sampling algorithm, and is thus more efficient than any D-ARM algorithm known today. Not only does this algorithm divide the disk-I/O costs of the single scan by partitioning the database among several machines, but also uses the combined memory to linearly increase the size of the sample.

This increase further improves the performance of the algorithm because the safety margin required in Sampling decreases when the (global) sample size increases. Extensive experiments on standard synthetic benchmarks show that D-Sampling is superior to previous algorithms in every way. When compared to Sampling one of the best sequential algorithms known today it offers super-linear speedup. When compared to FDM, it improves runtime by orders of magnitude. Finally, on scalability tests, an increase in both the number of computing nodes and the size of the database does not degrade D-Sampling performance. FDM, on the other hand, suffers performance degradation in these tests.

2. PREVIOUS WORK

Many algorithms, representing several different approaches, were suggested. Some algorithms, such as Apriori, Partition, DHP, DIC, and FP-growth, are bottom-up, starting from itemsets of size eand working up. Others, like Pincer-Search, use a hybrid approach, trying to guess large itemsets at an early stage. Most algorithms, including those cited above, adhere to the original problem definition, while others search for different kinds of rules. Algorithms for the D-ARM problem usually can be seen as parallelizations of sequential ARM algorithms. The CD, FDM, and DDM algorithms parallelize Apriori, and PDM parallelizes DHP . The major difference between parallel algorithms is in the architecture of the parallel machine. This may be shared memory, distributed shared memory, or shared nothing. One of the best sequential ARM algorithms. The idea behind Sampling is simple.

A random sample of the database is used to predict all the frequent itemsets, which are then validated in a single database scan. Because this approach is probabilistic, and therefore fallible, not only the frequent itemsets are counted in the scan but also their negative border. If the scan reveals that itemsets that were predicted to belong to the negative border are frequent then a second scan is performed to discover whether any superset of these itemsets is also frequent. To further reduce the chance of failure, Toivonen suggests that mining be performed using some *low_fr* < *MinFreq*, and the results reported only if they pass the original *MinFreq* threshold. He also gives a heuristic which can be used to determine the cost of using *low_fr* is an increase in the number of candidates.

The performance of the two is thus unrivaled by any other sequential ARM algorithm. The algorithm presented here combines ideas from several groups of algorithms. It first mines a sample of the database and then validates the result and can, thus, be seen as a parallelization of the Sampling algorithm. The sample is stored in a vertical trie structure that resembles the one in, and it is mined using modifications of the DDM algorithm, which is Apriori-based.

3. D-SAMPLING ALGORITHM

All distributed ARM algorithms that have been presented until now are Apriori based and thus require multiple database scans. The reason why no distributed form of Sampling was suggested in the six years since its presentation may lie in the communication complexity of the problem.
As we have seen, the communication complexity of D-ARM algorithms is highly dependent on the number of candidates and on the noise level in the partitioned database.

When Sampling reduces the database through sampling and lowers the threshold, it greatly increases both the number of candidates and the noise level. This may render a distributed algorithm useless. This is the reason that the reduced communication complexity of DDM seems to offer an opportunity. The main idea of D-Sampling is to utilize DDM to mine a distributed sample using *low_fr* .instead of *MinFreq*. After *low_fr* has been identified, the partitioned database is scanned once in parallel to find the actual frequencies of *low_fr* and its negative border. Those frequencies can then be collected and rules can be generated from itemsets more frequent than MinFreq.

Three modifications has been made to this scheme. First, although the given DDM is levelwise, here it is executed on a memory resident sample. Thus we could modify DDM to develop new itemsets on-the-fly and calculate their estimated frequency with no disk-I/O. Second, a new method for the reduction of *MinFreq* to *low_fr* yielded two additional benefits: it is not heuristic, i.e., our error bound is rigorous, and it produces many less candidates than the rigorous method suggested previously.

Third, after scanning the database, it would not be wise to just collect the frequencies of all candidates. Since these candidates were calculated according to the lowered threshold, few of them are expected to have frequencies above the original *MinFreq*. Instead, we run DDM once more to decide which candidates are frequent and which are not.

3.1 ALGORITHM

D-Sampling begins by loading a sample into memory. The sample is stored in a trie-a lexicographic tree. This trie is the main data structure of D-Sampling and is

accessed by all its subroutines. Each node of the trie stores, in addition to structural information (parents, descendants etc.), the list of TIDs of those transactions that include the itemset associated with this node. These lists are initialized from the sample for the first level of the trie; when a new trie node- and itemset - are developed, the TID lists of two of the parent nodes are intersected to create the TID list of the new node. The first step of D-Sampling is to run a modification of DDM on the distributed sample.

Then, in order to set *low_fr* the algorithm enters a loop; in each cycle through the loop it calls another DDM derivative called M-Max to mine the next M estimated-frequent itemsets. M ais a tunable parameter we set to about 100. After it finds those additional itemsets, D-Sampling reduces *low_fr* to the estimated frequency of the least frequent one and re-estimates the error probability using a formula described in section 4. When this probability drops below the required error probability, the loop ends. Then D-Sampling creates the final candidate set C by adding to *low_fr* its negative border.

ALGORITHM 1 D-SAMPLING

For node i out of n

Input:

MinFreq,MinConf,DB,s,M,δ

Output:

The set of confident association between globally frequent itemsets

Main:

Set p_error $\leftarrow$ 1, low_fr $\leftarrow$ MinFreq

Load a sample s^i of size s from DB^i into memory

Initialize the trie with all the size-1 itemset and calculate their TID lists

Low_fr $\leftarrow$ MDDM(MinFreq)

While p_error $\leftarrow$ δ

1.low_fr $\leftarrow$ low_fr U M_Max(M)

2.set low_fr to the frequency of least frequent itemset in low_fr

3.set p_error to the new error bound according to MinFreq, low_fr,F_{low_fr} .

Let C be F_{low_fr} U Negative_Border(F_{low_fr}) scan the database and compute Freq(C, DB^i) for each c£C. Update the frequency in the trie to the computed ones. Compute $f_{minFreq}$ such that c≠F_{low_fr} report failure Gen_Rules($F_{MinFreq}$,MinConf).

Once the candidate set is established, each partition of the database is scanned exactly once and in parallel, and the actual frequencies of each candidate are calculated. With these frequencies D-Sampling performs yet another round of the modified DDM. In this round the original *MinFreq* is used; thus, unless there is a failure, this round should never develop a candidate which is outside the negative border.

If indeed no failure occurs, then all frequent itemsets will be evaluated according to the actual frequencies which were found in the database scan. Hence, after this round it is known which of the candidates in yare globally frequent and which are not. In this case, rules are generated from $F_{MinFreq}$ using the known global frequencies.

If an itemset belonging to the negative border of F_{low_fr} does turn out to be frequent, this means that DSampling has failed: a superset of that candidate, which was not counted, might also turn out to be frequent.

In this case we suggest the same solution offered by Toivonen: to create a group of additional candidates that includes all combinations of anticipated and unanticipated frequent itemsets, and then perform an additional scan. The size of this group is limited by the number of anticipated frequent itemsets times the number of possible combinations of unanticipated frequent itemsets.

Since failures are very rare events, and the probability of multiple failure is exponentially small, the additional scan will incur costs that are of the same scale as the first scan.

3.2 MDDM - A MODIFIED DISTRIBUTED DECISION MINER

The original MDDM algorithm, as described in, is levelwise. When the database is small enough to fit into memory, the levelwise structure of the algorithm becomes super-fluous. Modified Distributed Decision Miner, or MDDM(Algorithm 2), therefore starts by developing all the locally frequent candidates, regardless of their size. It then continues to develop candidates whenever they are required, i.e., when all their subsets are assumed frequent (according tothe local hypothesis) or when another node refers to the associated itemset. The remaining steps in MDDM are the same as in DDM. Each party looks for itemsets for which the global hypothesis and local hypothesis disagree and communicate their local counts to the rest of the parties.

When no such itemsets exists, the party passes (it can return to activity if new information arrives). If all of the parties pass, the algorithm terminates and the itemsets which are predicted to be frequent according to the public hypothesis are the estimated globally frequent ones.

If a message is received for an itemset which has not yet been developed, it is developed on-the-fly and its local frequency is calculated.

ALGORITHM2MODIFIED DISTRIBUTED DECISION MINER

For node I out of n

Input:

fr-target frequency

Output:

F_{fr}

Definitions:

$$P(x,s^i) = \frac{\sum_{j\pounds G(X)}|s^j|Freq(x,s^j)}{|S|}$$

$$H(X)= \frac{\sum_{j\pounds G(X)}|s^j|freq(x,s^j)}{\sum_{j\pounds G(X)}|s^j|}$$

Main:

Develop all the candidates which are more frequent than *fr* according to P

Do

- Choose a candidate X that was not yet choosen and for which either $H(X) < fr \leq P(X,s^i)$ or $P(X,s^i) < fr \leq H(X)$
- Broadcast $m=<id(X), freq(X,S^i)>$
- If no such itemset exist broadcast <pass>

Untill |Passed|=N

R $\leftarrow$ all X with $H(X) \geq fr$

return R

when node i receive a message m from party j

1) If m=<pass>insert j into Passed
2) Else m = <id(X),Freq(x,s_j)>
3) If j ϵ Passed remove j from Passed

Insert j to G(X)

Recalculate H(X) and P(X,S_i)

3.3 M-MAX ALGORITHM

The modified DDM algorithm identifies all itemsets with frequency above *MinFreq*. D-Sampling, however, requires a further decrease in the frequency of itemsets which are included in the database scan. The reason for this, is that three parameters affect the chances for failure. These are the size of the sample N, the size of the negative border, and the estimated frequency of the least frequent candidate. The first parameter is given, the second is a rather arbitrary value which we can calculate or bound, and the last parameter is the one we can control. The frequency of the least frequent candidate can be controlled by reducing *low_fr*. However, this must be done with care: lowering the frequency threshold increases the number of candidates. This increase depends on the distribution of itemsets in the database and is therefore nondeterministic. The larger number of candidates affects the scan time: the more candidates you have, the more comparisons must be made per transaction. In a distributed setting, the number of candidates is also strongly tied to the communication complexity of the algorithm. To better control the reduction of *low_fr*, we propose another version of DDM called M-Max (Algorithm 3).

MMax increases the number of frequent itemsets by a given factor rather than decreasing the threshold value by an arbitrary value. Although worst case analysis shows that an increase of even one frequent itemset may require that any number of additional candidates be considered, the number of such candidates tends to remain small and roughly proportional to the number of additional frequent itemsets.

The M-Max algorithm is based on the inference that changing the *MinFreq* threshold to the H-value of the Ma-largest itemset1 every time an itemset is developed or a hypothesis value is changed will result in all parties agreeing on the a most frequent itemsets when DDM terminates. This is easy to prove. Take any final state of the modified algorithm. The H value of each itemset is equal in all parties; hence, the final *MinFreq* is equal in all parties as well.

ALGORITHM 3 M-MAX

For node I out of n

Input:

low_fr-target frequency

Output:

The M most frequent itemsets not yet in F_{low_fr}

Definitions:

Same as for MDDM algorithm

Let B denote theinitial size of F_{low_fr}, $fr=low\_fr$

Main:

Develop all the candidates which are more frequent than *fr* according to P

Do

- Choose a candidate X that was not yet choosen and for which either

$H(X) < fr \leq P(X,s^i)$ or $P(X,s^i) < fr \leq H(X)$

- Broadcast m=<id(X), freq(X,S^i)>
- If no such itemset exist broadcast <pass>

Untill |Passed|=N

R $\leftarrow$ all X in the trie with $H(X) \geq fr$ which are not in F_{low_fr}.

return R

when node i receive a message m from party j

1) If m=<pass>insert j into Passed
2) Else m = <id(X),Freq(x,s^j)>
3) If j ϵ Passed remove j from Passed

Insert j to G(X)

Recalculate H(X) and P(X,S^i)

Call *set_fr*

Procedure *set_fr:*

Do M times:

Select next most frequent itemset outside F_{low_fr} and develop its descendant if they have not been developed yet. Set *fr* to H value of last itemset selected. For itemset H=0 consider P instead.

The static *MinFreq* value set to the one finally agreed upon. The state attained by M-Max is also a valid final state for this DDM. Thus, by virtue of DDM correctness, all parties must be in agreement on the same set of frequent itemsets.

As a stand-alone ARM M-Max may be impractical because a node may be required to refer to itemsets it has not yet developed. If the database is large, this would require an additional disk scan whenever new candidates are developed. Nevertheless, at the *low_fr* correction stage of D-Sampling, the database is the memory-resident sample.

5. CONCLUSIONS AND FUTURE RESEARCH

We presented a new D-ARM algorithm that uses the communication efficiency of the DDM algorithm to parallelize the single-scan Sampling algorithm. Experiments prove that the new algorithm has superlinear speedup and outperforms both FDM and DDM with any *MinFreq* value. The exact improvement in relation to previous algorithms depends on the number of database scans they require.

Experiments demonstrate good scalability, provided the database scan is the major bottleneck of the algorithm. Some open questions still remain. First, it would be interesting to continue partitioning the database until every partition becomes memory resident. This approach may lead to a D-ARM algorithm that mines a database by loading it into the memory of large number of computers and then runs with no disk-I/O at all. Second, it would be interesting to have a parallelized version of the other single-scan ARM algorithm . DIC on a share-nothing cluster**,** or of the two-scans partition algorithm. Finally, the full potential of the M-Max algorithm has not yet been realized; we intend to research additional applications for this algorithm.

5. REFERENCES

[1] R. Agrawal, T. Imielinski, and A. N. Swami. Mining association rules between sets of items in large databases. In *Proc. of the 1993 ACM SIGMOD Int'l. Conference on Management of Data*, pages 207.216, Washington, D.C., June 1993.

[2] R. Agrawal and J. Shafer. Parallel mining of association rules. *IEEE Transactions on Knowledge and Data Engineering*, 8(6):962 . 969, 1996.

[3] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proc. of the 20th Int'l. Conference on Very Large Databases (VLDB'94)*, pages 487 . 499, Santiago, Chile, September 1994.

[4] V. S. Ananthanarayana, D. K. Subramanian, and M. N. Murty. Scalable, distributed and dynamic mining of association rules. In *Proceedings of HiPC'00*, pages 559.566, Bangalore, India, 2000.

[5] S. Brin, R. Motwani, J. Ullman, and S. Tsur. Dynamic itemset counting and implication rules for market basket data. *SIGMOD Record*, 6(2):255.264, June 1997.

[6] D. Cheung, J. Han, V. Ng, A. Fu, and Y. Fu. A fast distributed algorithm for mining association rules. In *Proc. Of 1996 Int'l. Conf. on Parallel and Distributed Information Systems*, pages 31 . 44, Miami Beach, Florida, December 1996.

[7] T. Hagerup and C. Rub. A guided tour of Chernoff bounds. *Information Processing Letters*, 33:305 . 308, 1989/90.

[8] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. Technical Report 99-12, Simon Fraser University, October 1999.

[9] Z. Jarai, A. Virmani, and L. Iftode. Towards a cost-effective parallel data mining approach. Workshop on High Performance Data Mining (held in conjunction with IPPS'98), March 1998.

[10] D.-I. Lin and Z. M. Kedem. Pincer search: A new algorithm for discovering the maximum frequent set. In *Extending Database Technology*, pages 105.119, 1998.